Extensible Name Service (XNS)™ Technical Specifications

Version 1.0

Published by the XNS Public Trust Organization (XNSORG)

July 9, 2002

Copyright

Document History

This is the first public release of the XNS Technical Specifications by the XNS Public Trust Organization (XNSORG). XNSORG is chartered to manage the future evolution of these pecifications in the public interest. XNSORG will always maintain a current version of these specifications at its web site at the DNS address: www.xns.org, XNS address: @XNSORG.

After an initial public review period, XNSORG anticipates delegating further work on the XNS Technical Specifications to an established technical standards body. To participate in the public review period or provide input regarding the choice of standards body, please join the mailing lists or comment on the public forums available at www.xns.org.

July 9, 2001 - Version 1.0

Initial publication of XNS Technical Specifications v1.0 by XNSORG.

Table of Contents

Chapter 2 XNS Base Services Specification (cont.)

Chapter 2 XNS Base Services Specification (cont.)

Chapter 3 XNS Addressing Specification 135

1

Introduction

This chapter introduces the Extensible Name Service (XNS), an open protocol for digital identity and relationship management. Additionally, the conventions and terminology used in this specification document are described.

Overview

Extensible Name Service (XNS) is a protocol for digital identity and relationship management that spans any number of devices and domains. Whereas DNS (Domain Name System) is a protocol designed for federated naming of Internet hosts at the TCP/IP level, XNS is designed for modeling and managing the identity of any actor at the SOAP [2] level, including people, businesses, machines, applications, objects, classes, etc. XNS enables identity controllers to register and use XNS identities to automate the exchange of any set of data associated with an identity while providing protection for the security and privacy of this data.

XNS is an open public set of technical specifications and a global community services infrastructure managed by the XNS Public Trust Organization (XNSORG), an independent international non-profit organization. XNSORG maintains the XNS Technical Specifications at the XNSORG web site at www.xns.org. XNS is based on the XML family of standards from the World Wide Web Consortium (W3C) and is freely available to all users and developers under a standard community license from XNSORG.

This document and its appendices are the normative technical specifications for XNS. The following companion documents, though non-normative, are recommended as for further understanding the requirements and architecture of XNS:

- *The Identity Web: Key Concepts of XNS Architecture* is a short overview to the major architectural features of XNS.
- *From Name Service to Identity Service: How XNS Builds on the DNS Model* is a detailed technical white paper that summarizes the differences in requirements and architecture between DNS and XNS.
- *XNS Use Cases* shows how XNS solves common problems of cross-domain identity management.
- *XNS Service Models* contains the abstract UML object models from which each of the concrete XNS service definitions are derived.

These documents may be obtained from the XNSORG Web site, www.xns.org.

Structure and Format of the Specifications

Like DNS, XNS is a globally distributed database. Unlike DNS, which is based on pre-defined resource records, in XNS each set of data representing an identity is logically represented as an XML document. This document, called the *identity document*, contains instances of XNS datatypes associated with that identity. The use of XML as the resource description format allows XNS to offer many of the self-description and extensibility advantages of Web services.

In addition to attributes associated with an identity, XNS identity documents also contain links to other identity documents the same way Web pages contain links to other Web pages. Called *identity links*, these fragments of the larger XML document contain references to the attributes shared with another identity together with the terms and conditions governing this data exchange relationship. This feature of XNS architecture permits the creation of identity webs the same way the Web specifications (HTML, HTTP, and URLs) enabled the creation of document webs.

An identity document is acted on by a software program called an *identity agent* to provide identity management and transaction services on behalf of the identity controller. Identity agents may operate on special servers, called *identity servers*, or they may be incorporated into other applications such as directory servers, email servers, contact managers, CRM applications, etc. Identity agents may also run on *identity clients*, e.g., edge devices such as smart cards, cell phones, smart phones, PDAs, laptops, etc.

Identity documents are managed by invoking a Web services interface (i.e., XML messaging) to the identity agent responsible for the document. XNS identities can invoke each other's services directly on a peer-to-peer basis, or they can be invoked by an external exposure such as a browser, an application, or another web service. A special set of XNS datatypes are used to create and store XNS datatype and message definitions. A logical collection of these definitions is called an *XNS service definition*. The XNS service definitions that comprise the base interoperability layer of XNS are called the *XNS base services*.

XNS currently uses Web Services Description Language (WSDL) Version 1.1 [3] as the normative format for XNS service definitions. WSDL is currently a W3C Note and is undergoing standardization by a Web Services Activity at the W3C. It is expected that the XNS specifications will maintain compatibility with the specifications developed by this Activity as well as other Web services definition languages or specifications developed by OASIS, IETF, and other Web services standards bodies as appropriate.

> **Note:** Because WSDL is a new specification that has not yet been standardized by a Web standards body, XNSORG may in the future adopt a different web service description format as the normative format for XNS services.

WSDL service definitions define the datatypes on which they operate using XML Schema Definition (XSD) language as specified by the W3C XML Schemas 1.0 Recommendation [4]. This is the normative schema definition format for XNS. A special XNS base service, Core, exists to define

the abstract schemas from which all concrete XNS objects and messages are derived. Core also defines the base XML datatypes used across all other XNS services (in addition to the simple datatypes defined in the W3C XML Schemas 1.0 Recommendation Part 2 [5]). Additional XSD files are used to define the concrete datatypes or message types used by a particular XNS service. The set of all WSDL and XSD files defining the XNS base services comprises the XNS Base Services Specification.

The ability to create persistent references across the globally distributed network of XNS identity documents requires an XML-based addressing syntax similar to the addressing portion of XPath. Like XPath, XNS addressing syntax is expressed in Extended Backus-Naur Form (EBNF) as defined in the W3C XML 1.0 Recommendation [6]. The components of XNS addressing syntax are based on a set of global addressing attributes defined in XNS Core Service. The validity and normalization rules for these attribute values are expressed in either EBNF or IETF RFC rule format as defined in IETF RFC-2119 [7]. Together these rule sets are captured in the XNS Addressing Specification.

The Role of XNS Discovery Service: Living Specifications

Like the XML Schema 1.0 specification, XNS is self-defining—it is described in terms of itself. Unlike the XML Schema specification, however, XNS service specifications are *dynamic*. Since they are published as XNS identity documents, XNS service specifications can be discovered, versioned, published, subscribed, and linked just like any other data in an identity document. This allows XNS to be a truly organic, self-extending system.

One of the XNS base services, Discovery, allows XNS identities to publish new XNS schema and service definitions and to retrieve the schema and service definitions published by other identities. Furthermore, since the industry standards for these definition formats are themselves constantly evolving, Discovery service supports the automated translation of any XNS schema or service definition into other industry standard formats as they appear. For example, as soon as the current W3C WSDL Note is standardized by the W3C Web Services Activity, XNSORG will be able to publish a service definition binding (see Future Work) and an updated enumeration of the WSDL service definition bindings in the XNS Discovery DescribeService message. As quickly as this updated service definition format is supported by at least one implementation, XNS developers will be able to get any XNS service definition in the most recent WSDL definition format.

The Role of XNS General Identities: A Shared Schema Ecology

XNS recognizes three fundamental types of identity controllers:

1. **Persons.** Any natural-born individual, regardless of their legal status, can be represented by one or more XNS *personal identities*.

2. **Organizations.** Any legally defined organizational entity (including sole proprietorships, partnerships, corporations, non-profits, governments, public host communities, academic institutions, etc.) can be represented by one or more XNS *organizational identities* (also called *business identities* for short).

3. **The general public.** Many objects in the world have identity besides people and organizations. These are the basis for generic nouns in the English language – objects such as the planets in the solar system, the elements in the atomic table, the colors in the spectrum, the notes in a musical scale—upon whose general identity we must all agree in order to communicate with each other. These identities are not controlled by any person or organization, but by linguistic, cultural, or scientific convention. In XNS this class of identities are represented by XNS *general identities* under the auspices of XNSORG.

These three archetypes of identity controllers form a natural evolutionary progression. New identities, such as new words in a language or the discovery of a new dinosaur, typically start with an individual person, then spread outward into groups or organizations. Eventually those with the broadest public usefulness become part of the cultural fabric of society and end out in our generic vocabulary.

XNS models this organic process by allowing all three types of identity controllers to create, publish, and subscribe to XNS schema and service definitions. Persons and organizations do this with XNS personal and organizational identities, respectively. XNSORG, representing the general public, is responsible for delegating to other academic, government, and industry standards bodies the job of defining XNS general identities that provide maximum benefit to the XNS community. Like a "survival of the fittest" for vocabularies, those schema and service definitions with the broadest public usefulness will eventually, like new words in a language, become part the XNS general vocabulary.

This process begins with the definition of the XNS global community identity—the identity managed by XNSORG that publishes the XNS base schema and service specifications in this document. As explained in the XNS Addressing Specification, this identity has the XNS address ":" (colon) and the XNS general name "+xns".

Documentation Conventions

The following conventions are used throughout the XNS Technical Specifications:

- The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” as used in this document are to be interpreted as described in IETF RFC-2119 [7].
- Formal requirements in these specifications that use these keywords will be shown in italic blocks prefixed by a heading providing the full name of the rule.
- EBNF productions use the EBNF syntax notation as described in the W3C XML 1.0 Recommendation [6]. They will also follow the shading and fragment naming rules above.
- All XNS addresses given as normative in this document will use the XNS native URI scheme, where “xns:” is the prefix (to be registered by XNSORG as a formal URI scheme). Examples: “xns:+xns/Core/Identity”, “xns:@Example/Sports”. Any address given in this format is equivalent to the same address given in the XNS URN scheme (“urn:xns:”) or the XNS HTTP scheme as discussed in the XNS Resource Identifier section of the XNS Addressing Specification.

Terminology and Abbreviations

Like many specifications, XNS requires precise terminology to describe the system it is specifying. This is particularly important in the field of digital identity, where metadata and “metaconcepts” proliferate. To facilitate this, these specifications include an XNS Glossary as Appendix C. The definitions in this glossary are normative to any XNS specification that uses these terms or abbreviations.

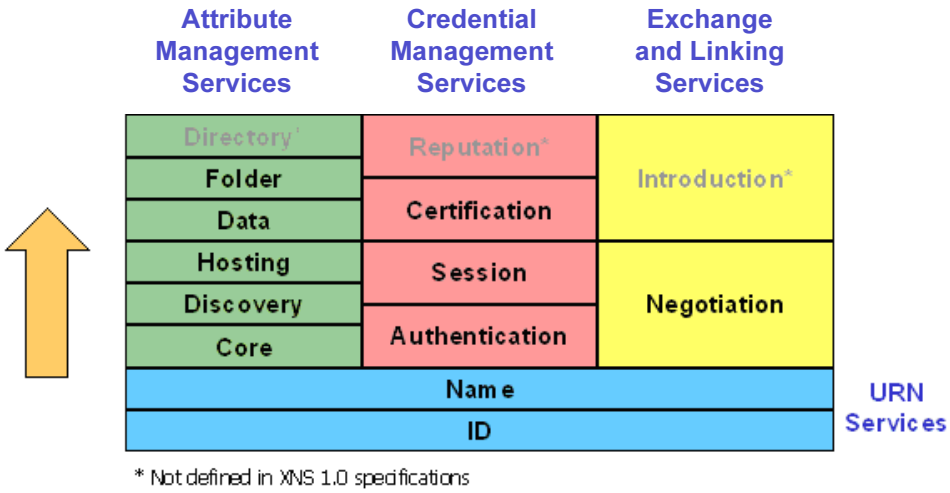
2

XNS Base Services Specification

This chapter contains an overview of the eleven services defined in the first release of the XNS specification, along with the additional services planned to be included in future versions of the specification.This chapter also includes human-readable service descriptions for each of the eleven services in XNS 1.0. Normative WSDL files for these services are in Appendix A, and normative XSD files for the related data types are in Appendix B.

Introduction to XNS Base Services

The following diagram provides a visual map of the XNS services and service categories (note that the categories are non-normative, since they serve only to group services by functional areas).



The XNS Base Services by functional category.

The following sections introduce the services in each service category.

URN Services

As described in the XNSORG technical white paper comparing XNS and DNS, two of the most fundamental requirements for a digital identity infrastructure are that it: a) enable an identity to be logically abstracted from any particular representation of that identity at a particular network location or device, and b) enable this logical abstraction to persist independent of changes to any concrete attribute of the identity.

Fulfilling this requirement requires both *semantic abstraction*—the separation of semantic identifiers (names) from persistent abstract identifiers (IDs)—and *identity federation*—the distributed resolution of names and IDs across a decentralized network of identity servers and clients. The result of these two architectural principles results in the need for what is technically known as a URN (Uniform Resource Name) service. More information about URN services is available through the IETF URN Charter. [1]

XNS implements an XML-based URN service via two XNS service definitions which provide the identity addressing foundation upon which all other XNS base services are built.

Service	Version	Description
Name	*1.0*	XNS names are federated, non-persistent, semantic identifiers optimized for human usability and memorability in referencing identity documents or attributes. XNS names may be modified or deleted at any time, and may also be transferred to other identity controllers, all without changing the underlying identity to which they resolve (established through ID Service, below). XNS Name service provides messages for registering, resolving, and reserving XNS names.
ID	*1.0*	XNS IDs are federated, persistent, non-semantic identifiers optimized for efficient machine resolution of references to and within identity documents. XNS IDs persist for the lifetime of an XNS resource and are never reused. XNS ID service enables an identity to register an XNS ID with another identity together with a list of URIs that allows the target identity to be physically located on the Internet. ID service also allows other identities to resolve an XNS ID to this list of URIs.

Attribute Management Services

A second fundamental requirement of XNS is that it permits any set of attributes to be associated with an identity. This need for extensibility in describing the actors participating in Web services mirrors the same requirement for XML itself. Rather than constantly update the definition of HTML to include new tags for new purposes, the W3C finally made the markup language itself extensible. This principle was further implemented with the addition of the W3C XML Schemas recommendation [4] which finally made XML documents fully self-describing.

Following this same design pattern, XNS architecture permits XNS identities to extend XNS by publishing their own XNS schema and service definitions. Once these definitions exist, they can be shared and linked across any population of identities, from two individuals to the entire XNS system. XNS schema and service definitions can also be composed modularly, so developers can reuse and extend datatypes or services already defined and deployed in the system (subject to the same data protection controls that exist for all XNS data). To do this, all references are based on persistent XNS identity addresses (URNs) so the references will not break if a resource moves on the network or a semantic name changes.

XNS attribute management is based on a *metaschema architecture* similar to XML itself. Thus it includes services for creating XNS schema and service definitions (Core and Discovery), plus services for managing instances of these definitions within and between identity documents (Hosting, Data, Folder, and Directory).

Service	Version	Description
Core	*1.0*	Analogous to XNS Schemas Part 2, XNS Core is the bootstrap service that defines the XNS abstract schemas and datatypes referenced by all other services, including the XNS abstract base message schema.
Discovery	*1.0*	Analogous to XNS Schemas Part 1, Discovery service defines XNS schema and service definitions and enables XNS identities to be queried for the schema and service definitions they publish or services they offer. It also allows schema and service definitions to be published in new and updated industry standard formats such as XSD and WSDL.
Hosting	*1.0*	Hosting service is the set of functions offered by a *host identity* to other XNS identities operating at the same network endpoint, called *hosted identities*. Hosting service standardizes the functions of creating and deleting XNS identity documents at any particular network location.
Data	*1.0*	Data service handles the basic functions of creating, reading, updating, and deleting attributes or attribute groups within an identity document–in essence, it defines the XML DOM (Document Object Model) interface for an identity. It also allows multiple updates to be performed in a single operation.
Folder	*1.0*	Folder service enables an identity controller to organize attributes and attribute groups in an identity document in the same way a computer file system owner organizes files. The identity controller can create, delete, or rename folders and subfolders and add, remove, or rename the data within a folder.
Directory	*Future*	Directory service will allow XNS identities to be organized in taxonomies in the same way Folder service allows attributes to be organized within an identity. Directory service will permit the creation of any number of taxonomies and ontologies by enabling XNS identities to: a) register with other XNS identities for directory entries based on their attributes, or b) search for other XNS identities matching an attribute query.

Credential Management Services

Another fundamental requirement of XNS is to allow every identity controller to protect the security and privacy of their identity data. This requires a common way of establishing and asserting identity credentials, i.e., the identity and attribute assertions needed to establish and maintain trust relationships. In XNS 1.0 these services are based primarily on the OASIS Security Assertion Markup Language (SAML) specifications [8].

Service	Version	Description
Authentication	*1.0*	Authentication service enables an identity controller to establish, manage, and validate the credentials necessary to prove they are the rightful controller of an XNS identity. Authentication service also handles application-level session management, i.e., assertions that the user provided the necessary credentials at a given time to create a session.
Session	*1.0*	Session service is a client/server service that allows browser-based authentication to be shared between many Web sites.
Certification	*1.0*	Certification service enables identities to request and receive digital certificates from each other. Certificates allow identities to make Boolean validity assertions regarding the attribute values or services of another identity, e.g., attest that a public key or a driver's license is a truthful representation. Certificates can also be revoked, with linked identities being notified.
Reputation	*Future*	Reputation service will allow XNS identities to make digitally signed value judgments regarding the attributes of another identity.

Exchange and Linking Services

Like DNS, XNS allows anyone to request the public attributes of an identity. However unlike DNS, XNS also supports the exchange of private, protected identity attributes—*identity transactions*—under the mutual consent and agreement of both parties. It further allows identities to form persistent links that model real-world trust relationships, including synchronization of shared identity attributes. These services are the very heart of the XNS protocol.

Service	Version	Description
Negotiation	*1.0*	Negotiation service is the basis for all private, protected data exchange between XNS identities. It allows identities to create, update, and delete links and contracts governing the identity attributes or services to be exchanged between the identities, including the applicable security, privacy, and synchronization permissions.
Introduction	*Future*	Introduction service will permit an identity linked to two other identities to introduce those two identities so they can form their own direct link. In essence it is a "three-way negotiation" service.

Detailed Service Descriptions

Each XNS service definition consists of a set of data schema definitions in XML Schema Definition 1.0 (XSD) format [4] and service definition in Web Services Description Language (WSDL) 1.1 format [3]. The special XNS schemas for constructing these definitions and the service definitions for reading them in industry-standard formats are defined in XNS Discovery Service.

Following are the detailed human-readable descriptions of each service and its component message and datatypes. This document also includes, for each service, a normative WSDL 1.1 definition file in Appendix A and a normative XSD 1.0 definition file in Appendix B.

Service: Core

+xns/Core

Contains data referenced by all services, as well as the abstract base message definition class.

Message Summary

Message	A communication sent to an identity and from an identity or an application.

Data Summary

Domain	Collection of identities that share a common legal owner.
DomainOwner	The legal owner of a domain.
Group	A collection of identities and other groups that have some aspect in common.
GroupOwner	The owner of the definition of a group.
INT	A non-negative integer.
IdentityDef	The set of attributes that describe and classify an identity.
IdentityTypeEnum	Categories by which an identity can be classified based on the nature of the principal it represents.
Membership	A pointer to a group to which an identity or another group belongs.
MsgStateEnum	Life-cycle states of a message.
Ref	A reference to an XML element within the identity document.
XLink	A data synchronization relationship between the containing XNS object and another XNS object.
XNSException	The details of a failed message.
XNSObject	A person, place, concept or thing that has attributes expressible in XML and that is addressable in XNS.

Datatype Summary

AbsoluteAddress	An XNS address that must be absolute but can be either an XNS ID or XNS name and can resolve to either an identity node or a data node within an identity.
ComplexType	Complex XML type as defined by the XML Schema specification.
DataID	An ID that is relative to the root node of an identity document.
DataIDNode	An XNS object in an XNS ID path that terminates in a node below the root node of an identity document.
DataName	A name that is relative to the root node of an identity document.
DataNameNode	An XNS object in an XNS name path that terminates in any node below the root node of an identity document.
HostID	The IdentityID of a host identity.
ID	A unique non-semantic identifier for a node in an identity document.
IdentityAddress	The address of an identity within the XNS network.
IdentityDataAddress	An absolute value that resolves to a lower-level node within an identity document.
IdentityDataID	An absolute value that resolves to a lower-level node within an identity document.
IdentityDataName	An absolute value that resolves to a lower-level node within an identity document.
IdentityID	A persistent globally unique identifier for an XNS identity.
IdentityIDNode	An XNS object in an XNS ID path that terminates in the root node of an identity document.
IdentityName	An absolute name for an XNS object at the root node of an identity document.
IdentityNameNode	An XNS object in an XNS name path that terminates in the root node of an identity document.
Name	A path that can begin with either 1) one of the three identity namespace prefix symbols, or 2) a double forward slash representing the abstract XNS name root identity managed by XNSORG.
NamespaceSymbol	A symbol used to indicate the three XNS-defined absolute namespaces.

Datatype Summary

OID	Object Identifier; one or more unique integers used in a dot-delimited path starting from a root node to form a unique ID for each object in the path.
RelativeDataID	An XNS ID that can only be resolved in a known context, i.e., from a specific node in an identity document.
RelativeDataName	An XNS name that can only be resolved in a known context, i.e., from a specific node in an identity document.
SimpleType	Simple XML type as defined by the XML Schema specification.
Text	Text in the form of a string.
URI	Uniform Resource Identifier.
URN	Uniform Resource Name as specified in IETF RFC 2141.
Version	A value used to specify the change history of an object.
VersionDate	A version value in the XML dateTime datatype format specified by W3C XML Schemas Part 2.
VersionNumber	An integer representing the version of the object.
XMLByteArray	An ordered arrangement of data elements.
XMLDateTime	A time instant as defined by the W3C XML schema specification.
XNSAddress	The top-level form of any address meeting the requirements of the XNS Addressing Specification.
XNSID	A permanent semantic identifier of any XNS object.
XNSName	A mutable semantic identifier of an XNS object.

Service Dependencies

+xns/Authentication
+xns/Certification

Message: Message

+xns/Core/Message
Derived from +xns/Core/XNSObject

A communication sent to an identity and from an identity or an application. This is an abstract base class for all XNS messages.

Input

Async Boolean	Is this message being sent asynchronously? True=message is asynchronous and sender is not waiting for a reply; False=message is synchronous and sender is awaiting a reply.
AuthSession *+xns/Authentication/AuthSession*	The authenticated session identifying the end-user who triggered the event resulting in the creation of this message.
Authorization *+xns/Certification/AuthorizationCert*	Message authorization provided by some identity. The identity receiving a message with an authorization may decide to perform its own authorization or use the authorizations provided in this element.
Date *+xns/Core/XMLDateTime*	Timestamp of when the message was constructed.
From *+xns/Core/IdentityAddress*	Address of the identity initiating the message. If the message is initiated from an application, then this is blank.
ID *+xns/Core/ID*	Global identifier of the message object.
Identity *+xns/Core/IdentityDef*	The portion of identity the from-identity wishes to provide to the to-identity for purposes of message authorization. Some messages may require that the from-identity posess specific identity attributes (such as being an identity of a certain type, or a member of a certain group). If the from-identity is willing to provide this information in order to receive the service, it is provided in this element.
MsgState *+xns/Core/MsgStateEnum*	The phase of a message's life cycle.

Input

RespondTo *+xns/Core/IdentityAddress*	Address of the identity to receive the response to this asynchronous message. This only applies if Async is True. If the message is asynchronous and RespondTo is blank, the response is not sent.
To *+xns/Core/IdentityAddress*	Address of the identity targeted by the message.
type *+xns/Core/IdentityDataAddress*	Address of the schema that defines this message.

Output

Date *+xns/Core/XMLDateTime*	Timestamp of when the message was constructed.
From *+xns/Core/IdentityAddress*	Address of the identity initiating the message. If the message is initiated from an application, then this is blank.
ID *+xns/Core/ID*	Global identifier of the message object.
MsgState *+xns/Core/MsgStateEnum*	The phase of a message's life cycle.
RespondTo *+xns/Core/IdentityAddress*	Address of the identity to receive the response to this asynchronous message. This only applies if Async is 'True.' If the message is asynchronous and RespondTo is blank, the response is not sent.
To *+xns/Core/IdentityAddress*	Address of the identity targeted by the message.
XNSException *+xns/Core/XNSException*	The details of a failed message. This is contained within a message object when the message state is EXCEPTION.
type *+xns/Core/IdentityDataAddress*	Address of the schema that defines this message.

Exceptions

InvalidArgument	A required input parameter was not supplied or an input parameter is of the incorrect type.

Data: Domain

+xns/Core/Domain
Derived from +xns/Core/Group

Collection of identities that share a common legal owner.

Data Elements

DomainOwner *+xns/Core/DomainOwner*	The owner of this domain.

Data: DomainOwner

+xns/Core/DomainOwner
Derived from +xns/Core/GroupOwner

The legal owner of a domain. The identity that contains this object is the owner of the referenced domain.

Data Elements

Domain *+xns/Core/Domain*	The domain owned by this domain owner.

Data: Group

+xns/Core/Group
Derived from +xns/Core/XNSObject

A collection of identities and other groups that have some aspect in common.

Data Elements

GroupOwner *+xns/Core/GroupOwner*	The owner of this group.
Groups *+xns/Core/IdentityDataAddressList*	Addresses of groups that are members of this group. *List.*
Members *+xns/Core/IdentityAddressList*	Addresses of identities that are members of this group. *List.*
Memberships *+xns/Core/MembershipList*	Memberships this group participates in. A membership specifies a group to which this group belongs. *List.*

Data: GroupOwner

+xns/Core/GroupOwner
Derived from +xns/Core/XNSObject

The owner of the definition of a group. The identity that contains this object is the owner of the referenced group.

Data Elements

Group *+xns/Core/Group*	The group owned by this group owner.

Data: INT

+xns/Core/INT
Derived from +xns/Core/XNSObject

A non-negative integer. See XNS Addressing Specification.

```
EBNF: INT ::= Non-negative integer
```

Data: IdentityDef

+xns/Core/INT
Derived from +xns/Core/XNSObject

The set of attributes that describe and classify an identity.

Data Elements

IdentityType *+xns/Core/IdentityTypeEnum*	The classification of an identity according to its basic nature of PER, ORG, or GEN.
Memberships *+xns/Core/MembershipList*	Memberships this identity participates in. A membership specifies a group to which this identity belongs. *List.*
PublicKey *+xns/Certification/PublicKeyCert*	The certified public key owned by this identity.
Types *+xns/Core/XNSObjectList*	XNS objects that contain attributes defining this identity. Each XNS object in the list is described by a type, providing the classification for the identity. An identity can be defined by multiple XNS objects. *List.*

Data: IdentityTypeEnum

+xns/Core/IdentityTypeEnum
Derived from +xns/Core/XNSObject

Categories by which an identity can be classified based on the nature of the principal it represents. The enumeration values include:

- **PER**–Personal identity.
- **ORG**–Organization or business identity.
- **GEN**–General identity.

Data: Membership

+xns/Core/Membership
Derived from +xns/Core/XNSObject

A pointer to a group to which an identity or another group belongs.

Data Elements

Group *+xns/Core/IdentityDataAddress*	Address of the group to which this membership applies.
GroupOwner *+xns/Core/IdentityAddress*	Address of the identity that owns the group to which this membership applies.
GroupType *+xns/Core/IdentityDataAddress*	Address of the schema that defines the specialization of the group to which this membership applies. If it is not specified, then the type is *+xns/Core/Group*.

Data: MsgStateEnum

+xns/Core/MsgStateEnum
Derived from +xns/Core/XNSObject

Life-cycle states of a message. The enumeration values include:

- **REQUEST**–The initial, imperative part of a message.
- **RESPONSE**–A response to a message, sent by the targeted identity.
- **EXCEPTION**–A notification of a failed message. If the message is in this state, the targeted identity is restored to its state prior to the message being processed.

Data: Ref

+xns/Core/Ref
Derived from +xns/Core/XNSObject

A reference to an XML element within the identity document. References may point to XNS objects within the identity document or to XML elements which are not top nodes of XNS objects.

Data: XLink

+xns/Core/XLink
Derived from +xns/Core/XNSObject

A data synchronization relationship between the containing XNS object and another XNS object.

Data Elements

Contract *+xns/Core/IdentityDataAddress*	Address of the XNS contract governing this link. This attribute must point to an XNS object of type +xns/Negotiation/Contract.
ID *+xns/Core/ID*	Immutable identifier of this link. ID must be unique in the ID namespace within which the link is contained.
Identity *+xns/Core/IdentityAddress*	Address of the identity that contains the XNS object to which the XNS object that contains this link is linked.
LastUpdate *+xns/Core/XMLDateTime*	Date this link was last updated. If this is an active link (i.e., push on change), then this attribute must not be present. In this case, the last update date is the last update date of the object the link is contained within. If this is not an active link (i.e., scheduled push, push on demand, etc.), then this attribute must be present, and it may or may not be the same date as the last update date of the object containing this link. The absense or presense of this attribute informs the identity to push on change or not (respectively).
Name String	Human-readable name for this element. Name must be unique in the namespace within which the element is contained. Although name can be used to identify an element, it is subject to change.

Data: XNSException

+xns/Core/XNSException
Derived from +xns/Core/XNSObject

The details of a failed message. This is contained within a message object when the message state is EXCEPTION.

Data Elements

Description String	Explanation of the error in terms useful to a programmer.
Name String	Textual identifier of the XNS exception.
XNSException *+xns/Core/XNSException*	A nested XNS exception. Exceptions contained within other exceptions provide finer-grained detail of the problem.

Data: XNSObject

+xns/Core/XNSObject

This is the base class from which all other XNS-addressable nodes are derived. It also contains the global addressing attributes used in the XNS Addressing Specification.

Data Elements

AV Boolean	Should automatic versioning be applied? True=all changes to this object result in a new version of the object; False=auto-versioning is not applied.
Description String	One-line summary description of the object.
IC Integer	ID counter containing the last ID assigned within the ID namespace. ID counter is maintained by the identity and is therefore read-only.
ID *+xns/Core/ID*	Immutable identifier of this object. ID must be unique in the ID namespace within which the object or reference is contained.

Data Elements

IName String	The independent name of this object. If an XNS object is requested through a reference, the independent name contains the name of the independent XNS object, while the Name attribute contains the reference name of the object. If an XNS object is requested directly, the Name attribute contains the name of the independent object and the independent name is blank.
IV Integer	Instance version. This is a sequential integer that reflects the version of this object.
Name String	Human-readable name for this object. Name must be unique in the namespace within which the object is contained. Although name can be used to identify an object, it is subject to change.
Notes String	User-entered textual information about this object.
PI Boolean	Are IDs in this ID namespace promoted to the parent ID namespace? This flag indicates whether or not this object is an ID namespace. True=this object is not an ID namespace—promote IDs of objects contained by this object to the parent ID namespace; False=this object is an ID namespace. All list type objects contain this attribute and only list type objects contain this attribute.
PN Boolean	Are names in this namespace promoted to the parent namespace? This flag indicates whether or not this object is a namespace. True=this object is not a namespace—promote names of objects contained by this object to the parent namespace; False=this object is a namespace.
Public Boolean	Is this object publically available? This flag indicates whether or not the data should be made available to any other identity upon request, without a contract. True=data is public; False=data is not public.

Data Elements

Ref *+xns/Core/XNSID*	Reference containing the identity-relative ID of the independent object. Ref is present only when this object is contained by reference.
SV Integer	Schema version; the version of the schema that defines this object. A new schema version is created when a change introduces an incompatibility with an earlier version. Programs use SV to detect the need to either upgrade the object to a later version or process the object as an earlier version.
XA *+xns/Core/XNSAddress*	Address of this object. XA is an independent address of an object (apart from any containment) relative to its identity. The identity-relative ID portion of the XA should be used as the persistent key in external references to the object because it never changes and is not dependent upon any container.
XNSBackRefs *+xns/Core/RefList*	IDs of XNS objects which have references to this object. This list is used to count the number of references so that this object will not be deleted if there are references. *List.*
XNSCerts *+xns/Certification/AttributeCertList*	Collection of digital certificates. The certificates provide external validation of the contents of this object. *List.*
XNSCreateDate *+xns/Core/XMLDateTime*	Date and time the object was first created. XNSCreateDate is created by the identity and is therefore read-only.
XNSLinks *+xns/Core/XLinkList*	Links to other XNS objects. Items in this list are used by the Data service to control synchronization of XNS objects. *List.*
XNSOnUpdate *+xns/Core/TextList*	Messages to send upon update of this object. XNSOnUpdate contains a list of message names along with their arguments. These messages will be created and executed when this object is updated. The message types included in this list must be derived from +xns/Data/RespondToUpdate.*List.*

Data Elements

XNSUpdateDate *+xns/Core/XMLDateTime*	Date and time the identity last modified the object.
XNSVersions *+xns/Core/RefList*	References to all prior versions of this object. *List.*
XNSXRefs *+xns/Core/XNSAddressList*	Addresses of equivalent objects. Equivalent objects are objects that represent the same thing in the real world.*List.*
type *+xns/Core/IdentityDataAddress*	Address of the schema that defines this object. This attribute can be included in any object, but must be included in a contained object if the contained object is lower in the class hierarchy than the container's definition of the object. This attribute must also be present if the object is new (i.e., has not had an ID assigned).

Datatype: AbsoluteAddress +xns/Core/AbsoluteAddress

An XNS address that must be absolute but can be either an XNS ID or XNS name and can resolve to either an identity node or a data node within an identity. See XNS Addressing Specification.

```
EBNF: AbsoluteAddress ::= IdentityAddress | IdentityDataAddress
```

Datatype: ComplexType +xns/Core/ComplexType

Complex XML type as defined by the XML Schema specification.

Datatype: DataID +xns/Core/DataID

An ID that is relative to the root node of an identity document. See XNS Addressing Specification.

```
EBNF: DataID ::= ';' DataIDNode [RelativeDataID]
```

Datatype: DataIDNode

+xns/Core/DataIDNode

An XNS object in an XNS ID path that terminates in a node below the root node of an identity document. See XNS Addressing Specification.

```
EBNF: DataIDNode ::= ID | ( '(' IdentityDataID ')' )
```

Datatype: DataName

+xns/Core/DataName

A name that is relative to the root node of an identity document. See XNS Addressing Specification.

```
EBNF: DataName ::= '/' RelativeDataName
```

Datatype: DataNameNode

+xns/Core/DataNameNode

An XNS object in an XNS name path that terminates in any node below the root node of an identity document. See XNS Addressing Specification.

```
EBNF: DataNameNode ::= Name | ( '(' IdentityDataAddress ')' )
```

Datatype: HostID

+xns/Core/HostID

The IdentityID of a host identity. A host identity is an identity that represents a network endpoint. Host identities provide XNS hosting service to other identities at the same network endpoint. See XNS Addressing Specification.

```
EBNF: HostID ::= ('(' URN ')' ['.' OID]) | OID
```

Datatype: ID

+xns/Core/ID

A unique non-semantic identifier for a node in an identity document. See XNS Addressing Specification.

```
EBNF: ID ::= XML character string normalized according to the XNS ID
Normalization Rules
```

Datatype: IdentityAddress

+xns/Core/IdentityAddress

The address of an identity within the XNS network. An identity address can consist of either an identity ID path or an identity name. The two can also be combined into a single addressing value. See XNS Addressing Specification.

```
EBNF: IdentityAddress ::= (IdentityID ['!' IdentityName]) | IdentityName
```

Datatype: IdentityDataAddress +xns/Core/IdentityDataAddress

An absolute value that resolves to a lower-level node within an identity document. See XNS Addressing Specification.

```
EBNF: IdentityDataAddress ::= (IdentityDataID ['!' IdentityDataName]) |
IdentityDataName
```

Datatype: IdentityDataID +xns/Core/IdentityDataID

An absolute value that resolves to a lower-level node within an identity document. See XNS Addressing Specification.

```
EBNF: IdentityDataID ::= IdentityID DataID
```

Datatype: IdentityDataName +xns/Core/IdentityDataName

An absolute value that resolves to a lower-level node within an identity document. See XNS Addressing Specification.

```
EBNF: IdentityDataName ::= IdentityName DataName
```

Datatype: IdentityID +xns/Core/IdentityID

A persistent globally unique identifier for an XNS identity. See XNS Addressing Specification.

```
EBNF: IdentityID = ':' HostID ':' [IdentityIDNode *('.' IdentityIDNode)]
```

Datatype: IdentityIDNode +xns/Core/IdentityIDNode

An XNS object in an XNS ID path that terminates in the root node of an identity document. See XNS Addressing Specification.

```
EBNF: IdentityIDNode ::= ID | '(' IdentityID ')'
```

Datatype: IdentityName +xns/Core/IdentityName

An absolute name for an XNS object at the root node of an identity document. See XNS Addressing Specification.

```
EBNF: IdentityName ::= (NamespaceSymbol | '//') IdentityNameNode *('/'
IdentityNameNode)
```

Datatype: IdentityNameNode +xns/Core/IdentityNameNode

An XNS object in an XNS name path that terminates in the root node of an identity document. See XNS Addressing Specification.

```
EBNF: IdentityNameNode ::= Name | '(' IdentityAddress ')'
```

Datatype: Name +xns/Core/Name

A path that can begin with either 1) one of the three identity namespace prefix symbols, or 2) a double forward slash representing the abstract XNS name root identity managed by XNSORG. See XNS Addressing Specification.

```
EBNF: Name ::= XML character string normalized according to the XNS Name
Normalization Rules
```

Datatype: NamespaceSymbol +xns/Core/NamespaceSymbol

A symbol used to indicate the three XNS-defined absolute namespaces. See XNS Addressing Specification.

```
EBNF: NamespaceSymbol ::= ('=' | '@' | '+' )
```

Datatype: OID +xns/Core/OID

Object Identifier; one or more unique integers used in a dot-delimited path starting from a root node to form a unique ID for each object in the path. See XNS Addressing Specification.

```
EBNF: OID ::= INT *('.' INT)
```

Datatype: RelativeDataID +xns/Core/RelativeDataID

An XNS ID that can only be resolved in a known context, i.e., from a specific node in an identity document. See XNS Addressing Specification.

```
EBNF: RelativeDataID ::= *('.' DataIDNode) [',' Version]
```

Datatype: RelativeDataName +xns/Core/RelativeDataName

An XNS name that can only be resolved in a known context, i.e., from a specific node in an identity document. See XNS Addressing Specification.

```
EBNF: RelativeDataName ::= DataNameNode *('/' DataNameNode) ['/,'
Version]
```

Datatype: SimpleType

+xns/Core/SimpleType

Simple XML type as defined by the XML Schema specification.

Datatype: Text

+xns/Core/Text

Text in the form of a string.

Datatype: URI

+xns/Core/URI

Uniform Resource Identifier. See the IETF URI Specification.

Datatype: URN

+xns/Core/URN

Uniform Resource Name as specified in IETF RFC 2141. See XNS Addressing Specification.

```
EBNF: URN ::= Uniform Resource Name as specified in IETF RFC 2141
```

Datatype: Version

+xns/Core/Version

A value used to specify the change history of an object. Versions can be in two formats: a) version numbers (integers) and b) version dates (expressed in the XML dateTime datatype format specified by W3C XML Schemas Part 2). See XNS Addressing Specification.

```
EBNF: Version ::= ('v' VersionNumber) | ('t' VersionDate)
```

Datatype: VersionDate

+xns/Core/VersionDate

A version value in the XML dateTime datatype format specified by W3C XML Schemas Part 2.See XNS Addressing Specification.

```
EBNF: VersionDate ::= XML DateTime instance
```

Datatype: VersionNumber

+xns/Core/VersionNumber

An integer representing the version of the object. See XNS Addressing Specification.

```
EBNF: VersionNumber ::= Non-negative integer
```

Datatype: XMLByteArray

+xns/Core/XMLByteArray

An ordered arrangement of data elements.

Datatype: XMLDateTime

+xns/Core/XMLDateTime

A time instant as defined by the W3C XML Schema specification.

Datatype: XNSAddress

+xns/Core/XNSAddress

The top-level form of any address meeting the requirements of the XNS Addressing Specification. An XNS address can be an XNSID, XNSName, or AbsoluteAddress. See XNS Addressing Specification.

```
EBNF: XNSAddress ::= XNSID | XNSName | AbsoluteAddress
```

Datatype: XNSID

+xns/Core/XNSID

A permanent semantic identifier of any XNS object. See XNS Addressing Specification.

```
EBNF: XNSID ::= IdentityID | IdentityDataID | DataID | RelativeDataID
```

Datatype: XNSName

+xns/Core/XNSName

A mutable semantic identifier of an XNS object. It can be one of four types depending on whether it is the absolute name for the XNS object at the root node of an identity document (IdentityName), the absolute name for an XNS object below the root node of an identity document (IdentityDataName), the relative name of an XNS object in relationship to the root node of an identity document (DataName), or the relative name for an XNS object in relationship to the current node of an identity document (RelativeDataName).

See XNS Addressing Specification.

```
EBNF: XNSName ::= IdentityName | IdentityDataName | DataName |
RelativeDataName
```

Service: Authentication

+xns/Authentication

Enables an indentity controller to validate that they are the rightful controller of an XNS identity. In addition, the authentication service provides session management for application interaction with the XNS identity. Session certification can be obtained providing evidence that the user provided the correct credentials at a given time to create a session.

Message Summary

CertifySession	Produces an authentication certificate for use by another identity.
ChangeCredentials	Changes authentication credentials, by replacing an existing credential with a new one, or deleting a credential, or adding a new credential.
GetWebLoginURI	Obtains the URI to redirect the browser to for identity authentication.
Login	Creates an application session, authenticating the application as being driven by the principal.
Logout	Invalidates an existing login session.

Data Summary

AuthSession	Authenticated session; a pointer to a session object that passes back to the client upon creation of the session object and that is included on subsequent messages from the client.
Credential	A set of data a login service can compare against a corresponding set of data on record to assert the identity of a particular user.
HashAlgorithmEnum	Valid hashing algorithms.
PasswordCredential	A type of credential consisting of user name and password combination.
PersonalIdentityCredential	An authentication credential created by a personal identity in an active login session for the purpose of utilizing that login session to authenticate to another identity.
Session	An authenticated application session.

Service Dependencies

+xns/Certification
+xns/Core

Message: CertifySession

+xns/Authentication/CertifySession
Derived from +xns/Core/Message

Produces an authentication certificate for use by another identity. This certificate may be used to provide single sign-on (or remote sign-on) for identities representing the same principal as the to-identity. It may also be used to authenticate to a non-personal identity.

- **To:** The identity an application is logged into.
- **From:** An application logged into an identity.

Input

Compress Boolean	Should the authentication certificate be compressed before encrypting? True=compress the authentication certificate after signing, but before encrypting; False=do not compress the authentication certificate.
EncryptFor *+xns/Core/IdentityAddress*	Address of the identity whose public key will be used to encrypt the authentication certificate. The authentication certificate created is usually used to authenticate to another identity. The host of the other identity is usually the identity doing the decryption, so it is common to encrypt the authentication certificate using the public key of the other identity's host. The authentication certificate can be encrypted for the other identity, but is less common because it requires more public keys to be known.
SignBy *+xns/Core/IdentityAddress*	Address of the identity requested to sign the certificate. This can be either the to-identity or the host of the to-identity. If the identity resides in an ID service provider with many other identities, it is more efficient for the host to sign the certificate because the receiver of the certificate is more likely to have the public key of the host than the public key of the to-identity in its cache.

Output

AuthCert String	Encrypted authentication certificate, Base64 encoded. The certificate has been signed, possibly compressed, then encrypted using the public key of the EncryptFor identity. If the SignBy identity supports the certification service, it may be used to verify the certificate. It may also be used to register for notification of session logout.

Message: ChangeCredentials

+xns/Authentication/ChangeCredentials
Derived from +xns/Core/Message

Changes authentication credentials, by replacing an existing credential with a new one, or deleting a credential, or adding a new credential.

- **To:** Any identity.
- **From:** Principal through an application.

Input

NewCredential *+xns/Authentication/Credential*	New credential to be added or to replace an old credential. If this argument is left blank, then the old credential will be deleted.
OldCredential *+xns/Authentication/Credential*	Credential to be replaced. This argument is left blank if adding a credential.

Exceptions

InvalidCredentials	Re-authentication credentials do not match credentials on file.

Message: GetWebLoginURI

+xns/Authentication/GetWebLoginURI
Derived from +xns/Core/Message

Obtains the URI to redirect the browser to for identity authentication. To use this URI, the browser must be redirected to this URI. The redirect must include the parameter "RedirectTo" containing the URI to redirect the user to following authentication. It can include the parameter "FailureTo" containing the URI to redirect the user to following authentication failure. It can also include the parameter "IdentityName" containing the identity name to authenticate. It can also include the parameter "IdentityID" containing the ID of the identity to authenticate.

- **To:** Identity to authenticate against.
- **From:** Web application wanting to redirect the user for authentication.

Output

WebLoginURI *+xns/Core/URI*	The URI to redirect the browser to for identity authentication.

Exceptions

NoUI	This login service does not provide a web user interface.

Message: Login

+xns/Authentication/Login
Derived from +xns/Core/Message

Creates an application session, authenticating the application as being driven by the principal. The session is created by the application providing an authentication credential obtained from the principal. Upon successful authentication, an authentication certificate is created, associated with the session, and provided in the return message. Authentication certificates are supplied on subsequent messages from the application and forwarded in messages to provide a context for the originator of a message chain.

- **To:** Identity to authenticate against.
- **From:** Principal through an application.

Input

Credentials *+xns/Authentication/CredentialList*	Proof of the principal's identity. *Required; List.*

Output

AuthSession *+xns/Authentication/AuthSession*	The authentication session associated with the application. This is attached to subsequent messages to the authenticated identity.
IdentityAddress *+xns/Core/IdentityAddress*	Resolved address of the authenticated identity.

Exceptions

InvalidCredentials	Credentials do not match credentials on file.

Message: Logout

+xns/Authentication/Logout
derived from +xns/Core/Message

Invalidates an existing login session.

- **To:** The identity an application is logged into.
- **From:** An application logged into an identity.

Input

AuthSession *+xns/Core/IdentityDataAddress*	Address of the authentication session associated with the application.

Data: AuthSession

+xns/Authentication/AuthSession
Derived from +xns/Core/XNSObject

Authenticated session; a pointer to a session object that passes back to the client upon creation of the session object and that is included on subsequent messages from the client. This is an abstract base class, because there can be many different types of authenticated sessions. Derived authenticated sessions must contain not only the authenticated session data itself, but enough information to uniquely identify the user within the login domain.

Data Elements

ConnectionID String	A unique value that binds the authenticated session to a specific instance of a connector.
SessionHandle String	A unique identifier that allows this authenticated session to be mapped to a session object stored in the server. The internal form of the session handle and the way the session handle is mapped to a session object is up to individual implementations, so a client application should not make assumptions about the internal semantics of this element.

Data: Credential

+xns/Authentication/Credential
Derived from +xns/Core/XNSObject

A set of data a login service can compare against a corresponding set of data on record to assert the identity of a particular user. This is an abstract base class, because there can be many different types of credentials. Derived credentials must contain not only the credential data itself, but enough information to uniquely identify the user within the login domain.

Data: HashAlgorithmEnum

+xns/Authentication/HashAlgorithmEnum
Derived from +xns/Core/XNSObject

Valid hashing algorithms. The enumeration values include:

- **MD5:** Message Digest 5.
- **NONE:** Plain text.

Data: PasswordCredential

+xns/Authentication/PasswordCredential
Derived from +xns/Authentication/Credential

A type of credential consisting of user name and password combination.

Data Elements

HashAlgorithm *+xns/Authentication/HashAlgorithmEnum*	The hashing algorithm used to encrypt the password credential.
Password String	A secret textual string known only to the user.
UserName String	A textual string that uniquely identifies the user within the login domain.

Data: PersonalIdentityCredential

+xns/Authentication/PersonalIdentityCredential
Derived from +xns/Authentication/Credential

An authentication credential created by a personal identity in an active login session for the purpose of utilizing that login session to authenticate to another identity. The identity this credential will be used for has a trust relationship with the personal identity, allowing it to be used as an authentication identity. The personal identity may be a web-based personal identity or a hardware-based identity, such as a smart card, capable of producing and signing authentication certificates.

Data Elements

AuthCert String	Encrypted authentication certificate, Base64 encoded. This certificate was created by the +xns/Authentication/CertifySession message for the purpose of certifying a login session with the specified personal identity. The authentication certificate has been signed, possibly compressed, then encrypted using the public key of the EncryptedFor identity. If the SignedBy identity supports the certification service, it may be used to verify the authentication certificate. It may also be used to register for notification of session logout.
Compressed Boolean	Is the authentication certificate compressed? True=the authentication certificate was compressed prior to encryption, and must be uncompressed after decryption; False=the authentication certificate is not compressed.

Data Elements

EncryptedFor *+xns/Core/IdentityAddress*	Address of the identity owning the public key used to encrypt the authentication certificate. This will be either the identity which the personal identity credential will be used to authenticate against or that identity's host. It will usually be encrypted for the host, thus reducing the number of public keys required to be known.
PersonalIdentity *+xns/Core/IdentityAddress*	Address of the personal identity the credential represents.
SignedBy *+xns/Core/IdentityAddress*	Address of the identity who signed the contained authentication certificate. This can be either the specified personal identity or the host of the personal identity. Most authentication certificates are signed by the host in order to reduce the number of public keys required to be known.

Data: Session

+xns/Authentication/Session
Derived from +xns/Core/XNSObject

An authenticated application session. This object is created upon a successful login. It represents the status of the authentication session.

Data Elements

AuthCert *+xns/Certification/AuthenticationCert*	An authentication certificate created upon login. This certificate is not signed because it is either used as a template for an authentication certificate produced by the *+xns/Authentication/CertifySession* message or used in a message for identity-to-identity communication. If another authentication certificate is created from this one, the new authentication certificate will be signed. If the authentication certificate is passed in an identity-to-identity message, then the authentication certificate does not need to be signed because the entire message will either be signed (or not) based on the security parameters of the two identities.
IdentityAddress *+xns/Core/IdentityAddress*	Address of the identity that established the session. Address of the identity that established the session.

Service: Certification

+xns/Certification

Enables an identity to issue a certificate attesting that some object (e.g., a driver's license) is a truthful representation. Certificates can be revoked, with interested identities being notified. Certificates are signed and validated using public key infrastructure (PKI). This service also generates and publishes the public key for an identity, and can notify other identities if the public key changes.

Message Summary

Authorize	Authorizes a message to run.
ConfirmAttrCert	Confirms or denies certification, sent when a pending certification is finalized.
ConfirmRevocation	Notifies an identity who has registered for revocation notification that a certificate has been revoked.
GenerateKeyPair	Generates a new public/private key pair and returns a public key certificate.
GetCACerts	Retrieves all the CA certificates of the specified certificate type.
GetCRL	Retrieves the identity's current Certificate Revocation List (CRL).
GetPublicKeys	Retrieves the public key certificates for an XNS identity.
RequestAttrCert	Requests an attribute certificate for the specified object.
RevokeCert	Revokes a certificate issued by this identity and notifies all identities previously registered to be notified upon revocation.
SubmitForAttrCertConfirm	Allows the certifying identity to sign the certificate.
VerifyCert	Confirms validity of a certificate and optionally requests notification of revocation.
VerifySignature	Ensures that the data being verified has not changed.

Data Summary

AttributeCert	A certificate that makes reference to the XNS object being certified.
AuthenticationCert	A certificate signifying that the principal represented by an identity has been authenticated to that identity.

Data Summary

AuthorizationCert	A certificate signifying that the principal has been authorized to execute a message.
CACert	Assertion that the identity who has this object has been certified to be a certifying authority (CA) for the represented certificate type.
CRL	Certificate Revocation List; a list of certificates, by type, which this identity has created and subsequently revoked.
Certificate	An XNS digital certificate.
CredentialCert	A certificate created by a credential collector representing the contained credentials as an authentication attempt.
PublicKeyCert	An identity's certified public key.
SAMLAssertion	A SAML assertion; a package of information that supplies one or more statements made by an issuer.
SAMLAttributeAssertion	A SAML assertion containing an attribute statement.
SAMLAttributeStatement	A SAML attribute statement; a statement that asserts that the specified subject is associated with the supplied attributes.
SAMLAuthenticationAssertion	A SAML assertion containing an authentication statement.
SAMLAuthenticationStatement	A SAML authentication statement; a statement that asserts that the specified subject was authenticated by a particular means at a particular time.
SAMLAuthorizationDecisionAssertion	A SAML assertion containing an authorization decision statement.
SAMLAuthorizationDecisionStatement	A SAML authorization decision statement; a statement that asserts that a request to allow the specified subject to access the specified resource has been granted or denied.

Data Summary

SAMLSubjectAssertion	A SAML assertion containing a subject statement.
SAMLSubjectStatement	A SAML subject statement.
TrustedIdentity	An identity that is directly or indirectly trusted by the identity holding this object, along with specification of what it is trused to certify.

Datatype Summary

Signature	A digital signature as defined by the XML DSIG specification.

Service Dependencies

+xns/Authentication
+xns/Core

Message: Authorize

+xns/Certification/Authorize
Derived from +xns/Core/Message

Authorizes a message to run. A message (the provided message) is included as an input parameter in this message (the authorize message). The identity receiving the authorize message grants or denies authorization to run the provided message. If the provided message is authorized, it will be returned along with an authorization certificate testifying that this identity authorized the message to run. When the provided message is sent, the to-identity of the provided message will use this information to help it decide if it allows the message to run. If the to-identity of the provided message is the same as the to-identity of the authorize message, it is likely that the provided message will pass authorization when run. Because of changes that may occur between execution of the authorize message and sending the provided message, there is no guarantee that the provided message will run even if authorized.

Output

Message *+xns/Core/Message*	Message to authorize. Upon successful return, this message will contain an authorization certificate.

Exceptions

AuthDenied	Authorization attempt failed; reason is supplied in the text of the exception.

Message: ConfirmAttrCert

+xns/Certification/ConfirmAttrCert
Derived from +xns/Core/Message

Confirms or denies certification, sent when a pending certification is finalized. If the original +xns/Certification/RequestAttrCert message returns the pending flag as true, then the object is queued up for manual verification. Once verified, the originating identity receives this message confirming (or denying) the certification.

- **To:** Identity that owns the object for which certification is requested.
- **From:** Identity certifying the object.

Input

Cert *+xns/Certification/AttributeCert*	Requested attribute certificate. If the Certified parameter is set to 'True,' then this parameter contains the certificate generated for the specified object.
Certified Boolean	Was the object certified? True=certified; False=not certified.
Exception *+xns/Core/XNSException*	The reason the object could not be certified, if that is the case.

Message: ConfirmRevocation

+xns/Certification/ConfirmRevocation
Derived from +xns/Core/Message

Notifies an identity who has registered for revocation notification that a certificate has been revoked.

- **To:** Identity who requested revocation notification via the +xns/Certification/VerifyCert message.
- **From:** Identity requested to provide notification of revocation.

Input

CertificateID *+xns/Core/IdentityDataID*	Address of the certificate that has been revoked. *Required.*
Reason String	Reason the certificate was revoked.

Message: GenerateKeyPair

+xns/Certification/GenerateKeyPair
Derived from +xns/Core/Message

Generates a new public/private key pair and returns a public key certificate. The private key is stored within the identity. The public key certificate returned is signed by the identity's host. This message will succeed only if no key pair exists for the identity or the current public key certificate has been revoked (usually due to a compromised private key in the identity or somewhere in the CA key chain).

- **To:** Identity being requested to generate its new key pair.
- **From:** Any identity or application; ususally the host identity.

Output

PublicKey *+xns/Certification/PublicKeyCert*	Newly generated public key, certified by the to-identity's host.

Exceptions

CurrentKeyExpired	The public key certificate has expired, but has not been revoked. (Use RequestCert to generate a new certifiate with a new expiration date.)

Message: GetCACerts

+xns/Certification/GetCACerts
Derived from +xns/Core/Message

Retrieves all the CA certificates of the specified certificate type.

- **To:** Certifying identity.
- **From:** Any identity who wishes to confirm that the certifying identity is certified to certify.

Input

CertType *+xns/Core/IdentityDataAddress*	Address of the definition of the certificate type for which the CA certificates are requested. This is an address of an object of type *+xns/Discovery/DataTypeDef* containing the definition of a class derived from *+xns/Certification/Certificate*.
DataType *+xns/Core/IdentityDataAddress*	Address of a DataTypeDef object which defines the attribute certificate data type for which the CA certificates are requested. Data type is applicable for attribute certificates only (certificate type of *+xns/Certification/AttributeCert*) and defines the data type the to-identity is certified to certify. *Required.*

Output

CACerts *+xns/Certification/CACertList*	The CA certificates carried of the specified certificate type. *List.*

Message: GetCRL

+xns/Certification/GetCRL
Derived from +xns/Core/Message

Retrieves the identity's current Certificate Revocation List (CRL).

- **To:** CA identity or identity subscribing to the CA identity's CRL.
- **From:** Any identity or application that wants to keep a CRL for validity checking rather than making individual calls to +xns/Certification/VerifyCert.

Input

CAIdentity *+xns/Core/IdentityAddress*	Address of certificate authority identity whose CRL is requested.
CertType *+xns/Core/IdentityDataAddress*	Address of the definition of the certificate type for which the CRL is requested. This is an address of an object of type +xns/Discovery/DataTypeDef containing the definition of a class derived from +xns/Certification/Certificate.
DataType *+xns/Core/IdentityDataAddress*	Address of a DataTypeDef object that defines the data type of attribute certificates included in this CRL. Data type is applicable for attribute certificates only (certificate type of +xns/Certification/AttributeCert). If this data type is a base class, then the CRL including attribute certificates for all object types that inherit from this class would be returned. For example, if *+xns/Core/XNSObject* is specified, the CRL for all object types this identity has certified would be returned.

Output

CRL *+xns/Certification/CRL*	Current CRL object for the specified certificate type and specified CA identity.

Message: GetPublicKeys

+xns/Certification/GetPublicKeys
Derived from +xns/Core/Message

Retrieves the public key certificates for an XNS identity.

- **To:** Usually the identity for which public keys are requested, but can be any other identity that has knowledge of the list.
- **From:** Any identity or application requesting the public keys.

Input

Identity *+xns/Core/IdentityAddress*	Address of the identity for which public keys are requested.

Output

PublicKeys *+xns/Certification/PublicKeyCertList*	All public key certificates for this identity. *List.*

Message: RequestAttrCert

+xns/Certification/RequestAttrCert
Derived from +xns/Core/Message

Requests an attribute certificate for the specified object. If the certificate is to remain valid following changes to the object, the specified object should be versioned. The certificate will not apply to future versions of the object, but it will remain valid for this version. If the object is not versioned, any changes to the object will render the certificate invalid.

- **To:** Identity acting as certification authority.
- **From:** Any identity or application requesting certification of an object.

Input

ToCertify *+xns/Core/XNSObject*	Object to be certified. *Required.*

Output

Cert *+xns/Certification/AttributeCert*	Requested attribute certificate. If the Pending parameter is set to 'False,' then this parameter contains the certificate generated for the specified input object.
Pending Boolean	Is object certification pending? True=object could not be certified immediately and notification of certification will be sent at a later time via a +xns/Certification/ConfirmAttrCert message; False=object was certified and is attached.

Exceptions

Cannot Certify	Cannot certify this object for the reason contained in exception.

Message: RevokeCert

+xns/Certification/RevokeCert
Derived from +xns/Core/Message

Revokes a certificate issued by this identity and notifies all identities previously registered to be notified upon revocation.

- **To:** Identity who issued the certificate to be revoked.
- **From:** Any identity or application requesting revocation; usually the application authenticated against the to-identity.

Input

CertificateID *+xns/Core/IdentityDataID*	Address of the certificate to be revoked. *Required.*
Reason String	Reason for the revocation.

Exceptions

AlreadyRevoked	This certificate ID has already been revoked.

Message: SubmitForAttrCertConfirm

+xns/Certification/SubmitForAttrCertConfirm
Derived from +xns/Core/Message

Allows the certifying identity to sign the certificate. If the original +xns/Certification/RequestAttrCert message returns the Pending parameter as 'True,' then the object is queued up for manual verification. Once verified, this message allows the certifying identity to prepare the certificate. Then, the +xns/Certification/ConfirmAttrCert message is called to confirm (or deny) the certification.

- **To:** Identity certifying the object.
- **From:** Identity certifying the object.

Input

Cert *+xns/Certification/AttributeCert*	Requested attribute certificate. If the Certified parameter is set to 'True,' then this parameter contains the certificate generated for the specified object.
Certified Boolean	Was the object certified? True=the object was certified; False=the object was not certified.
Exception *+xns/Core/XNSException*	The reason the object could not be certified, if that is the case.

Message: VerifyCert

+xns/Certification/VerifyCert
Derived from +xns/Core/Message

Confirms validity of a certificate and optionally requests notification of revocation.

- **To:** Identity that either issued certificate or is capable of verifying the certificate.
- **From:** Any identity or application.

Input

CertificateID *+xns/Core/IdentityDataID*	Address of certificate to be verified.
NotifyUponRevocation *+xns/Core/IdentityAddress*	Address of identity to notify if and when this certificate is revoked.

Exceptions

RevocationNotificationNotPossible	Certificate is currently valid, but the to-identity is incapable of issuing or unwilling to issue revocation notification to the NotifyUponRevocation identity.

Message: VerifySignature

+xns/Certification/VerifySignature
Derived from +xns/Core/Message

Ensures that the data being verified has not changed. Specific algorithms are applied depending upon the type of certificate passed in.

- **To:** Identity that is capable of verifying the signature.
- **From:** Any identity or application.

Input

Certificate *+xns/Certification/Certificate*	The certificate containing the signature to be verified.

Data: AttributeCert

+xns/Certification/AttributeCert
Derived from +xns/Certification/Certificate

A certificate that makes reference to the XNS object being certified.

Data Elements

Assertion *+xns/Certification/SAMLAttributeAssertion*	The SAML attribute assertion.
Certifying *+xns/Core/IdentityDataID*	Address of the object being certified. This includes the identity identifier as well as the object identifier within the identity.

Data: AuthenticationCert

+xns/Certification/AuthenticationCert
Derived from +xns/Certification/Certificate

A certificate signifying that the principal represented by an identity has been authenticated to that identity.

Data Elements

Assertion *+xns/Certification/SAMLAuthenticationAssertion*	The SAML authentication assertion.

Data: AuthorizationCert

+xns/Certification/AuthorizationCert
Derived from +xns/Certification/Certificate

A certificate signifying that the principal has been authorized to execute a message.

Data Elements

Assertion *+xns/Certification/SAMLAuthorizationDecisionAssertion*	The SAML authorization decision assertion.

Data: CACert

+xns/Certification/CACert
Derived from +xns/Core/XNSobject

Assertion that the identity who has this object has been certified to be a certifying authority (CA) for the represented certificate type. While not itself a certificate, this object is a statement that the owner of the object is acting as a CA for the specified data type. The expectation is that this object contains at least one certificate of type *+xns/Certification/AttributeCert,* created by an authority that certifies this CA to be a CA for the specified data type.

Data Elements

CertType *+xns/Core/IdentityDataAddress*	Address of the definition of the certificate type to which this CA certificate applies. This is an address of an object of type *+xns/Discovery/DataTypeDef* containing the definition of a class derived from *+xns/Certification/Certificate.*
DataType *+xns/Core/IdentityDataAddress*	Address of a DataTypeDef object which defines the attribute certificate data type to which this CA certificate applies. Data type is applicable for attribute certificates only (certificate type of *+xns/Certification/AttributeCert).*

Data: CRL

+xns/Certification/CRL
Derived from +xns/Core/XNSobject

Certificate Revocation List; a list of certificates, by type, which this identity has created and subsequently revoked. Revoked certificates beyond their expiration date may not be included in this list.

Data Elements

CertType *+xns/Core/IdentityDataAddress*	Address of the definition of the certificate type contained in this CRL. This is an address of an object of type *+xns/Discovery/DataTypeDef* containing the definition of a class derived from *+xns/Certification/Certificate.*
DataType *+xns/Core/IdentityDataAddress*	Address of a DataTypeDef object that defines the data type of attribute certificates included in this CRL. Data type is applicable for attribute certificates only (certificate type of *+xns/Certification/AttributeCert).*
Revoked *+xns/Core/DataIDList*	Addresses referencing certificates which have been revoked. List.

Data: Certificate

+xns/Certification/Certificate
Derived from +xns/Core/XNSobject

An XNS digital certificate. This is an abstract base class from which all concrete certificate classes are derived. There are two types of concrete certificate classes. Instances of the first type, the attribute certificate class, reference the XNS object being certified. The second type includes all other certificate derivatives, each of which defines its own unique set of data being certified. Generally, attribute certificates certify other identity's data whereas all other certificate derivatives certify the data contained by the certificate.

Data Elements

NotifyOnRevocation *+xns/Core/IdentityAddressList*	List of addresses of identities to notify upon revocation. If an identity requests notification of revocation via the *+xns/Certification/VerifyCert* message, the address of that identity will be added to this list. If the certificate is revoked by the CA, the CA will send the *+xns/Certification/ConfirmRevocation* message to each identity in this list. List.
RevocationDate *+xns/Core/XMLDateTime*	Date of certificate revocation.If the certificate was revoked by the CA, it may contain this element specifying the date it was revoked. The presence of this element can be positive evidence of a revoked certificate, but the absence of this element is not positive evidence that the certificate is still valid. If positive evidence is needed, the message *+xns/Certification/VerifyCert* must be sent to the CA, the owner of this certificate.
RevocationReason String	Reason for certificate revocation. If the certificate was revoked by the CA, it may contain this element specifying the reason the CA revoked the certificate.

Data: CredentialCert

+xns/Certification/CredentialCert
Derived from +xns/Certification/Certificate

A certificate created by a credential collector representing the contained credentials as an authentication attempt. The credential collector has collected these credentials, but has not stored them, and is merely passing them on to an authentication service. By signing this certificate, the credential collector agrees to be bound by the XNS terms for credential collection.

Data Elements

Credential *+xns/Authentication/Credential*	The authentication credential.
DateCollected *+xns/Core/XMLDateTime*	Date the credential was collected. By creating this certificate, the credential collector represents this date to be the date the credentials were collected from the user. Security policies at the authentication authority may use this date to decide if they are willing to create an authentication certificate based on these credentials.

Data: PublicKeyCert

+xns/Certification/PublicKeyCert
Derived from +xns/Certification/Certificate

An identity's certified public key. The actual public key is contained in this object as an X.509 public key certificate.

Data Elements

Key String	The X.509 public key certificate. This is contained in the object in X.509 binary form, Base64 encoded.

Data: SAMLAssertion

+xns/Certification/SAMLAssertion
Derived from +xns/Core/XNSObject

A SAML assertion; a package of information that supplies one or more statements made by an issuer. See specification at www.oasis-open.org. In our implementation, the SAML assertion is abstract; more specific assertion classes are derived from this class. The concepts underlying these more specific classes are discussed in the SAML specification.

Data Elements

Audience *+xns/Core/URI*	A URI that identifies an intended audience for this assertion.
Issuer String	Address of the identity that issued the assertion. If this is a SAML assertion issued by XNS, the address should be of type *+xns/Core/IdentityAddress*.
NotBefore *+xns/Core/XMLDateTime*	Time instant at which the validity interval begins.
NotOnOrAfter *+xns/Core/XMLDateTime*	Time instant at which the validity interval ends.

Data: SAMLAttributeAssertion

+xns/Certification/SAMLAttributeAssertion
Derived from +xns/Certification/SAMLAssertion

A SAML assertion containing an attribute statement. See concept of this derived class in specification at www.oasis-open.org.

Data Elements

AttributeStatements *+xns/Certification/SAMLAttributeStatementList*	The SAML attribute statements represented in this SAML attribute assertion. *List*.

Data: SAMLAttributeStatement

+xns/Certification/SAMLAttributeStatement
Derived from +xns/Certification/SAMLSubjectStatement

A SAML attribute statement; a statement that asserts that the specified subject is associated with the supplied attributes. See specification at www.oasis-open.org.

Data: SAMLAuthenticationAssertion +xns/Certification/SAMLAuthenticationAssertion

Derived from +xns/Certification/SAMLAssertion

A SAML assertion containing an authentication statement. See concept of this derived class in specification at www.oasis-open.org.

Data Elements

AuthenticationStatements *+xns/Certification/SAMLAuthenticationStatementList*	The SAML authentication statements represented in this SAML authentication assertion. *List.*

Data: SAMLAuthenticationStatement

+xns/Certification/SAMLAuthenticationStatement

Derived from +xns/Certification/SAMLSubjectStatement

A SAML authentication statement; a statement that asserts that the specified subject was authenticated by a particular means at a particular time. See specification at www.oasis-open.org.

Data Elements

AuthenticationInstant *+xns/Core/XMLDateTime*	Time at which the authentication took place.
AuthenticationMethod *+xns/Core/URI*	A URI that specifies the type of authentication that took place.
DNSAddress String	The DNS address of the system entity that was authenticated.
IPAddress String	The IP address of the system entity that was authenticated.

Data: SAMLAuthorizationDecisionAssertion

+xns/Certification/SAMLAuthorizationDecisionAssertion
Derived from +xns/Certification/SAMLAssertion

A SAML assertion containing an authorization decision statement. See concept of this derived class in specification at www.oasis-open.org.

Data Elements

AuthorizationDecisionStatements *+xns/Certification/SAMLAuthorizationDecisionStatementList*	The SAML authorization decision statements represented in this SAML authorization decision assertion. *List.*

Data: SAMLAuthorizationDecisionStatement

+xns/Certification/SAMLAuthorizationDecisionStatement
Derived from +xns/Certification/SAMLSubjectStatement

A SAML authorization decision statement; a statement that asserts that a request to allow the specified subject to access the specified resource has been granted or denied. See specification at www.oasis-open.org.

Data: SAMLSubjectAssertion

+xns/Certification/SAMLSubjectAssertion
Derived from +xns/Certification/SAMLAssertion

A SAML assertion containing a subject statement. See concept of this derived class in specification at www.oasis-open.org.

Data Elements

SubjectStatements *+xns/Certification/SAMLSubjectStatementList*	The SAML subject statements represented in this SAML subject assertion. *List.*

Data: SAMLSubjectStatement

+xns/Certification/SAMLSubjectStatement
Derived from +xns/Core/XNSObject

A SAML subject statement. This is a base class that serves as an extension point as well as the class from which all other SAML statements are derived. See specification at www.oasis-open.org.

Data Elements

SecurityDomain String	The security domain governing the name of the subject.
SubjectName String	The name of the principal that is the subject of the statement.

Datatype: Signature

+xns/Certification/Signature

A digital signature as defined by the XML DSIG specification.

See http://www.w3.org/TR/xmldsig-core/.

Service: Data

+xns/Data

Manages attributes and attribute groups within an identity. Data within an identity can be added, deleted, or updated. Data of a given type can be retrieved. In addition, multiple updates to the data can be performed in a single operation.

Message Summary

Add	Adds an instance of a data object to an identity.
ChangeNotify	Notifies an identity that the underlying data source has changed.
Delete	Removes a reference to an object.
Get	Retrieves data of a specified type at a specified set of addresses from within identity documents.
GetListElements	Retrieves the elements of a list.
GetVersions	Returns a list of versions of a data object.
Increment	Increments an integer element within an object.
RespondToUpdate	Performs an action in response to modification of an XNS object.
Set	Performs multiple specified data changes to the identity.
Update	Modifies an element or attribute of an existing data object at the specified address.
UpdateObject	Updates the object at the specified address with the object provided.

Data Summary

ChangeRequest	A unit of work constituting a change to an identity's data.
Dir	A container for XNS objects and other directories that enables a user to organize his data.
Identity	Container for the principal's data; the identity document.
TypeCollection	A container for XNS objects of the same data type.

Service Dependencies

+xns/Core

Message: Add

+xns/Data/Add
Derived from +xns/Core/Message

Adds an instance of a data object to an identity.

This message adds the specified data object to the identity and returns the data path of the newly added object. The 'parent' argument points to a list of objects of the same type to which the data object can be added. The 'before' argument can be specified to add the object before this object in the list.

- **To:** Identity to whom the data object will be added.
- **From:** Identity or exposure providing the data object to be added.

Input

Before *+xns/Core/DataID*	Address of data object in list before which this object is to be added.
ContractID String	Identifier of the XNS contract governing the change. If the data change request is originated from an XNS contract, then the from-identity is the data provider, and this element identifies the contract authorizing the provider to make this update.
Object *+xns/Core/XNSObject*	Data object to be added. *Required.*
Parent *+xns/Core/DataID*	Address of list of objects to which the data object is to be added.

Output

DataPath *+xns/Core/DataID*	Data path of the newly added object.

Exceptions

BeforeNotFound	Data object before which this data object is to be added cannot be found.

Message: ChangeNotify

+xns/Data/ChangeNotify
Derived from +xns/Core/Message

Notifies an identity that the underlying data source has changed. If the identity is connected to an underlying data source that is capable of changing using a non-XNS interface, this is the mechanism for the underlying data source to notify the identity that its data has changed. It must do this to assure the identity has fresh data in its cache and to synchronize these changes with other XNS objects.

- **To:** Identity that owns the data at the specified address.
- **From:** Usually the to-identity or the to-identity's host.

Input

Changes *+xns/Core/DataIDList*	XNS object IDs representing the data objects that have changed. *List.*

Exceptions

InvalidID	One or more of the specified IDs are not represented in this identity.

Message: Delete

+xns/Data/Delete
Derived from +xns/Core/Message

Removes a reference to an object. If the last reference to this object within the identity is removed, then the object itself will be deleted.

- **To:** Identity from whom the reference to the object will be removed.
- **From:** Identity requesting the removal.

Input

ContractID String	Identifier of the XNS contract governing the change. If the data change request is originated from an XNS contract, then the from-identity is the data provider, and this element identifies the contract authorizing the provider to make this update.
DataAddress *+xns/Core/XNSAddress*	Address of the object reference to be deleted. *Required.*

Message: Get

+xns/Data/Get
Derived from +xns/Core/Message

Retrieves data of a specified type at a specified set of addresses from within identity documents.

An input data type is optional. If the data type is not specified, the data as type *+xns/Core/XNSObject* is returned.

The elements of the Addresses argument are XNS addresses. If an address does not specify an identity, the identity the message is sent to is assumed. A set of input addresses is optional. If no addresses are provided, then all instances of the specified data type from within the to-identity are returned.

Depth indicates how deep to traverse the XML. If the actual type of the object is derived from the specified type, the actual type will be returned in the 'type=' attribute of the objects.

- **To:** Identity supplying the data, either directly or via its contracts with other identities.
- **From:** Identity requesting data.

Input

Addresses *+xns/Core/TextList*	XNS addresses from which data is to be returned. *List.*
Depth Integer	Indicator of how deep to traverse the XML. If depth=0, then the elements of the object, but not of contained objects, are returned. If depth=-1, then all the data, as deep as exists, are returned.
IncludeTypes Boolean	Should the type attribute be set on returned elements? True=the type attribute is set on all elements returned; False=the type attribute is set on the requested elements only if their data type does not match exactly the data type requested.
Limit Integer	The maximum number of objects to be returned.
Skip Integer	The number of objects to be skipped when satisfying the get request.
Type String	The data type of the object to be created upon message return. This is a string in the format *+xns/Core/XNSObject.* If the data type is not known, a data type of *+xns/Core/XNSObject* can be specified; in this case, the actual data type of the derived object can be found in the 'type' attribute of *+xns/Core/XNSObject.*

Output

Data *+xns/Core/ElementList*	The data returned. *List.*

Exceptions

IncorrectType	An object in the Addresses list was not of the type specified by the Type input parameter.

Message: GetListElements

+xns/Data/GetListElements
Derived from +xns/Core/Message

Retrieves the elements of a list.

- **To:** Identity supplying the data, either directly or via its contracts with other identities.
- **From:** Identity requesting data.

Input

Address *+xns/Core/XNSAddress*	Address of the list from which elements are to be retrieved.
Depth Integer	Indicator of how deep to traverse the XML. If depth=-1, then all the data, as deep as exists, are returned.
IncludeTypes Boolean	Should the type attribute be set on returned elements? True=the type attribute is set on all elements returned; False=the type attribute is set on the requested elements only if their data type does not match exactly the data type rquested.
Limit Integer	The maximum number of elements from the list to be returned.
Skip Integer	The number of elements to be skipped when satisfying the get list elements request.
Type String	The data type of the object to be created upon message return. This is a string in the format *+xns/Core/XNSObject*. If the data type is not known, a data type of *+xns/Core/XNSObject* can be specified; in this case, the actual data type of the derived object can be found in the 'type' attribute.

Output

Data *+xns/Core/ElementList*	The data returned. *List*.

Exceptions

DataSetTooLarge	The number of elements in the list is too large to handle.

Message: GetVersions

+xns/Data/GetVersions
Derived from +xns/Core/Message

Returns a list of versions of a data object. The list is sorted in reverse date order, so that the most recent version is first. If there are no stable versions, an empty list is returned.

- **To:** Identity that owns the data at the specified address.
- **From:** Usually the to-identity or the to-identity's host.

Input

Address *+xns/Core/XNSAddress*	The address of the object for which versions are requested.

Output

Versions *+xns/Core/RefList*	References to versions of the object, sorted in reverse chronological order. *List.*

Message: Increment

+xns/Data/Increment
Derived from +xns/Core/Message

Increments an integer element within an object.

- **To:** Identity containing the element to be incremented.
- **From:** Identity requesting the change.

Input

Address *+xns/Core/XNSAddress*	Address of the object containing the element to be incremented.
Amount Integer	The quantity by which the element is to be incremented.
ElementName String	The name of the element to be incremented.

Output

IncrementedValue Integer	The new value of the element after being incremented.

Message: RespondToUpdate

+xns/Data/RespondToUpdate
Derived from +xns/Core/Message

Performs an action in response to modification of an XNS object.

This is an abstract base class containing behavior to be inherited by messages that perform specific actions in response to specific object updates. Derivations of this message are run asynchronously and all exceptions and return arguments are discarded.

- **To:** Identity that owns the updated data.
- **From:** Identity that owns the updated data.

Input

Address *+xns/Core/IdentityDataID*	The absolute address of the updated object. *Required.*
Args *+xns/Core/TextList*	Arguments specified on the 'XNSOnUpdate' line of the updated object. If the 'XNSOnUpdate' line is: *@Identity/Service/Message arg1 arg2 arg3* ...then there would be three elements in this list: “arg1”, “arg2”, and “arg3”. *List.*
DataType *+xns/Core/IdentityDataAddress*	Address of a DataTypeDef object which defines the data type of the updated object. *Required.*

Message: Set

+xns/Data/Set
Derived from +xns/Core/Message

Performs multiple specified data changes to an identity.

This message accepts a list of changes (add, update, and/or delete) and applies all of them to the identity in a single operation. If no exceptions are raised, then all changes are applied. If an exception is raised, then no changes are applied.

If there are any "add" requests, then the data paths of the newly added objects are returned in the NewDataPaths element, in the order of the "add" requests.

- **To:** Identity to have changes applied.
- **From:** Identity requesting changes.

Input

Changes *+xns/Data/ChangeRequestList*	The changes to be applied. *Required; List.*
ContractID String	Identifier of the XNS contract governing the change. If the data change request is originated from an XNS contract, then the from-identity is the data provider, and this element identifies the contract authorizing the provider to make this update.

Output

NewDataPaths *+xns/Core/TextList*	Data paths of newly added objects. *List.*

Message: Update

+xns/Data/Update
Derived from +xns/Core/Message

Modifies an element or attribute of an existing data object at the specified address.

If the Value argument is left empty, then the element or attribute will be removed. If AttrName is specified, then the value is assigned to the attribute. If ElemName is specified, then the value is assigned to the element.

- **To:** Identity containing the element or attribute to be updated.
- **From:** Identity or exposure requesting the update and containing the new values.

Input

AttrName String	Name of the attribute to be updated.
ContractID String	Identifier of the XNS contract governing the change. If the data change request is originated from an XNS contract, then the from-identity is the data provider, and this element identifies the contract authorizing the provider to make this update.
DataPath *+xns/Core/DataID*	Data path of the element or attribute to be updated. Required.
ElemName String	Name of the data element to be updated.
Value String	Contents of the attribute or data element after update.

Exceptions

ElemNotFound	Element not found.

Message: UpdateObject

+xns/Data/UpdateObject
Derived from +xns/Core/Message

Updates the object at the specified address with the object provided. This message may be called instead of the *+xns/Data/Set* message to update an existing object in one of the following scenarios:

1) There is no way of knowing the prior state, therefore no way of specifying changes to that state.
2) Regardless of the prior state, the object must be placed in the specified state.

This message should be used only if the Set or individual Add/Update/Delete messages cannot be used, because it may result in undesirable side-effects in a multi-user environment. Specifically, if two applications change different elements of the same object at the same time, the Set message will allow both changes to take place, while the UpdateObject message may overwrite the first application's changes. On the other hand, if this is the desired effect, then the UpdateObject message may be more appropriate than Set.

- **To:** Identity in which to set the provided object.
- **From:** Usually the owner or distributor of data at the specified address.

Input

Address *+xns/Core/IdentityDataAddress*	Address of the object to update. This may be an address of an object owned by the to-identity, or it may be an address of data the to-identity has subscribed from another identity.
Object *+xns/Core/XNSObject*	Object to set at the specified address.
UpdateObject String	Identifier of the XNS contract governing the change. If the data change request is originated from an XNS contract, then the from-identity is the data provider, and this element identifies the contract authorizing the provider to make this update.

Exceptions

IncorrectType	The object at the specified address is not of the same type as the object provided.

Data: ChangeRequest

+xns/Data/ChangeRequest
Derived from +xns/Core/XNSObject

A unit of work constituting a change to an identity's data. Change request represents an object addition, deletion, or modification.

Data Elements

Address *+xns/Core/DataID*	Address of the data being changed.
Attr String	Name of the attribute to be updated or deleted.
Before *+xns/Core/DataID*	Address of the object which this object is to be inserted before. This is specified only for the addition of an object to a list. If a 'before' address is not specified and the parent is a list, then the object is added at the end of the list.
Object *+xns/Core/XNSObject*	The new object being added.
Parent *+xns/Core/DataID*	Address of the parent of the object being added.
Tag String	XML tag name of the element to be updated or deleted.
Type String	Change request type of “Add”, “Update”, or “Delete”. If it contains “Add”, then the object is to be added to the specified parent. If it contains “Update”, then the data element at the specified address and of the specified attribute or element name is updated. If it contains “Delete”, then the object, element, or attribute at the specified address and of the specified attribute or element name is to be deleted. If an element name or attribute name is not given, the entire object at the address is deleted.
Value String	The new value of the attribute or element. This is provided for the type “Update” only.

Data: Dir

+xns/Data/Dir
Derived from +xns/Core/XNSObject

A directory: a container for XNS objects and other directories for organizing user data.

Data Elements

Data *+xns/Core/RefList*	References to objects contained in this directory. *List*.
Dirs *+xns/Data/DirList*	Directories contained in this directory. *List*.

Data: Identity

+xns/Data/Identity
Derived from +xns/Core/XNSObject

Container for the principal's data; the identity document.

Data Elements

Data *+xns/Data/TypeCollectionList*	Repository of the identity's objects.
Dir *+xns/Data/Dir*	The top level directory for hierarchical organization of data.

Data: TypeCollection

+xns/Data/TypeCollection
Derived from +xns/Core/XNSObject

A container for XNS objects of the same data type. Data is organized within an identity document using type collections.

Data Elements

Derivatives *+xns/Core/IdentityDataAddressList*	List of data types known to inherit from the data type of this type collection.
DerivedFrom *+xns/Core/IdentityDataAddress*	Address of a DataTypeDef object which defines the data type that this type collection's data type is derived from.
Instances *+xns/Core/ElementList*	List of XNS objects contained within this type collection.
SchemaAddress *+xns/Core/IdentityDataAddress*	Address of a DataTypeDef object that defines the data type of objects contained in this type collection.
TypeName String	The XML tag name of the objects stored within the 'instances' element.
Versions *+xns/Core/ElementList*	List of previous independent versions of XNS objects contained within this type collection.

Service: Discovery

+xns/Discovery

Defines XNS schema and service definitions and enables XNS identities to be queried for the schema and service definitions they publish or services they offer. It also allows schema and service definitions to be published in new and updated industry standard formats such as XSD and WSDL. For example, as soon as the current W3C WSDL Note is standardized by the W3C Web Services Working Group, XNSORG will be able to publish an updated enumeration of the WSDL service definition formats in the GetServiceDefinition message. As quickly as this updated service definition format is supported by at least one implementation, XNS developers will be able to get any XNS service definition in this new format.

Message Summary

DescribeService	Provides a description of the specified service and links to optional supporting files.
GetServiceProviders	Provides a list of identities capable of providing a requested service on behalf of the to-identity.
GetServicesDefined	Produces a list of services that an identity has defined.
GetServicesOffered	Produces a list of services this identity offers.
RegisterServiceProvider	Registers services provided by an extended identity.
RetireServiceProvider	Retires services offered by a service provider.

Data Summary

DataElemDef	A description of an atomic piece of data; a component of a DataTypeDef object.
DataTypeDef	A description of an XNS object held by an identity.
Dependency	A relationship between this service (the dependent service) and another service (the independent service) that this service references.
EnumDef	Definition of an enumeration value.
MessageDef	A description of the structure of a communication sent or received by an identity.
ServiceDef	Definition of an XNS service, including messages, data, and dependencies on other services.

Service Dependencies

+xns/Core

Message: DescribeService

+xns/Discovery/DescribeService
Derived from +xns/Core/Message

Provides a description of the specified service and links to optional supporting files.

- **To:** Identity containing the service definition.
- **From:** Identity or exposure requesting the service definition.

Input

ServiceName *String*	Name of the service to be described. Text may be obtained from the +xns/Discovery/GetServicesDefined message.

Output

HTMLLink *+xns/Core/URI*	Link to the HTML documentation.
ServiceDef *+xns/Discovery/ServiceDef*	Service definition as an XNS object.
WSDLLink *+xns/Core/URI*	Link to the WSDL representation.
XSDLink *+xns/Core/URI*	Link to the XML schema specification.

Message: GetServiceProviders

+xns/Discovery/GetServiceProviders
Derived from +xns/Core/Message

Provides a list of identities capable of providing a requested service on behalf of the to-identity. The list of identities returned must represent the same identity as the to-identity.

- **To:** Any identity.
- **From:** Identity or exposure requesting the service providers.

Input

Service *+xns/Core/IdentityDataAddress*	Name of the service being requested, expressed as a data address.

Output

Providers *+xns/Core/IdentityAddressList*	Addresses of identities capable of providing this service. The list is ordered by priority. List.

Message: GetServicesDefined

+xns/Discovery/GetServicesDefined
Derived from +xns/Core/Message

Produces a list of services that an identity has defined.

- **To:** Identity containing the service definitions.
- **From:** Identity requesting the list of services defined.

Output

Services *+xns/Core/TextList*	Service names this identity has defined. These strings may be used in the ServiceName argument of the +xns/Discovery/DescribeService message. *List*.

Message: GetServicesOffered

+xns/Discovery/GetServicesOffered
Derived from +xns/Core/Message

Produces a list of services this identity offers.

This message produces three lists of services it offers, tailored to the requestor, i.e., the from-identity. The Public list includes all services it offers to the public. The Authorized list includes the services that the from-identity has available to it based upon prior negotiations. The'Negotiable list includes services this identity would offer upon successful negotiation of terms. A list of forms containing the terms for negotiable services is specified in the Terms list. Services specified in these three lists are fully qualified (e.g., *+xns/ID/Register).*

- **To:** Identity offering the services.
- **From:** Identity requesting to know what services are being offered.

Output

AuthorizedServices *+xns/Core/IdentityDataAddressList*	Services offered to the from-identity, expressed as addresses. *List.*
NegotiableServices *+xns/Core/IdentityDataAddressList*	Services the identity would offer upon successful negotiation of terms, expressed as addresses. The form containing the terms required for each service is in the corresponding list, Terms. Each service in the NegotiableServices list has a corresponding item in the Terms list. If no terms are required for negotiation, then a null string appears in the Terms list. *List.*
PublicServices *+xns/Core/IdentityDataAddressList*	Public services offered by this identity, expressed as addresses. *List.*
Terms *+xns/Core/TextList*	Forms containing the terms required to negotiate a service listed in NegotiableServices. Each form in this list corresponds to a service in NegotiableServices. If a service in NegotiableServices does not require terms, then a null string is included in Terms. *List.*

Message: RegisterServiceProvider

+xns/Discovery/RegisterServiceProvider
Derived from +xns/Core/Message

Registers services provided by an extended identity. This message associates the service provider as an identity agent capable of providing the specified services on behalf of the to-identity. The service provider must have an extension relationship with the to-identity (i.e., the service provider must represent the same identity as the to-identity).

- **To:** Any extended identity of the service provider.
- **From:** Either the service provider, or an identity acting on behalf of the service provider (such as its host).

Input

ServiceProvider *+xns/Core/IdentityAddress*	Address of the (extended) identity providing the service for this identity.
Services *+xns/Core/IdentityDataAddressList*	The addresses of the services being registered. *List.*

Message: RetireServiceProvider

+xns/Discovery/RegisterServiceProvider
Derived from +xns/Core/Message

Retires services offered by a service provider. See +xns/Discovery/RegisterServiceProvider.

- **To:** Any extended identity of the service provider.
- **From:** Either the service provider, or an identity acting on behalf of the service provider (such as its host).

Input

ServiceProvider *+xns/Core/IdentityAddress*	Address of the (extended) identity retiring from providing the service for this identity.
Services *+xns/Core/IdentityDataAddressList*	Addresses of the services being retired. List.

Data: DataElemDef

+xns/Discovery/DataElemDef
Derived from +xns/Core/XNSObject

A description of an atomic piece of data; a component of a DataTypeDef object.

Data Elements

DataType String	Name of the data type. The contents can be either a well-known datatype or a string in the format of an IdentityDataAddress *(+xns/Core/Identity-DataAddress)* which resolves to a DataTypeDef object. The well-known datatypes are: "String", "Integer", "Boolean", "TimeInstant", or any other core datatype specified as "DataType: *+xns/Core/type*" where type is a core XNS datatype. *Required.*
PromoteNames Boolean	Should names of objects contained in this list share the name space of the list? True=names of instances of objects in the list are promoted to the list's name space; False=the list itself is the name space. PromoteNames can be set only if Recurring is set to true in this object (i.e., this object defines a list).
Recurring Boolean	Is this a recurring data element? True=this element defines a list of objects; False=this element defines a single object or element.
Required Boolean	Is this data element required?

Data: DataTypeDef

+xns/Discovery/DataTypeDef
Derived from +xns/Core/XNSObject

A description of an XNS object held by an identity. A DataTypeDef object can contain either DataElemDef objects or EnumDef objects. If it contains DataElemDef objects, then the data type describes an XNS object. If it contains EnumDef objects, then the data type describes an enumeration.

Data Elements

DerivedFrom *+xns/Core/IdentityDataAddress*	Address of a DataTypeDef object that defines the data type that this type is derived from.
ElemDefs *+xns/Discovery/DataElemDefList*	Data element definitions. *List.*
EnumDefs *+xns/Discovery/EnumDefList*	Enumeration definitions. *List.*
TypeCollectionAddress *+xns/Core/IdentityDataAddress*	Address of a type collection that functions as the prototypical type collection for this data type. All type collections of the same type are logically equivalent and can be cross referenced to each other via the XNSXRefs element of XNSObject. The type collection to which this address points is the original type collection to which any other may be cross referenced.
Version Integer	Current compatibility version of this schema. A new schema version is created when a change introduces an incompatibility with an earlier version. Instances of this data type store Version in the SV attribute of XNSObject. Programs use SV to detect the need to either upgrade the object to a later version or process the object as an earlier version.

Data: Dependency

+xns/Discovery/Dependency
Derived from +xns/Core/XNSObject

A relationship between this service (the dependent service) and another service (the independent service) that this service references. An instance of a dependency is not required for the relationship between this service and the Core service, because dependency upon the Core service is assumed.

Data Elements

Identity *+xns/Core/IdentityAddress*	The address of the identity that defines the independent service.
Required Boolean	Is this dependency required? True=this service requires access to an implementation of the independent service; False=this service receives optional value if it has access to an implementation of the independent service.
ServiceName String	The name of the independent service.

Data: EnumDef

+xns/Discovery/EnumDef
Derived from +xns/Core/XNSObject

Definition of an enumeration value. This is one value of an enumeration object, which is a static list of valid strings that an element of an XNS object can contain.

Data Elements

IsDefault Boolean	Is this value the default enumeration value? True=this is the default value; False=this is not the default value.
SequenceNo Integer	Sequence number indicating the order in which the enumeration values appear.
Value String	The valid string comprising an enumeration value.

Data: MessageDef

+xns/Discovery/MessageDef
Derived from +xns/Core/XNSObject

A description of the structure of a communication sent or received by an identity.

Data Elements

Args *+xns/Discovery/DataElemDefList*	Message calling arguments; the calling data that is passed by the message. *List.*
DerivedFrom *+xns/Core/IdentityDataAddress*	Address of a MessageDef object that defines the message type that this message is derived from.
Exceptions *+xns/Core/XNSObjectList*	Known exceptions that may be thrown if the message implementation fails. *List.*
PostConditions String	Human readable description of the state of data returned if no exceptions are encountered.
PreConditions String	Human readable description of the required state of input data. Message implementations assume this state to be true and are not required to validate it. If the data is not in this state, the implementation may or may not throw an exception and the post condition is not guaranteed.
Return *+xns/Discovery/DataElemDefList*	Data that the message will return. *List.*

Data: ServiceDef

+xns/Discovery/ServiceDef
Derived from +xns/Core/XNSObject

Definition of an XNS service, including messages, data, and dependencies on other services.

Data Elements

Data *+xns/Discovery/DataTypeDefList*	The data types defined by the service. *List.*
Dependencies *+xns/Discovery/DependencyList*	Dependencies on other services. Dependencies describe other services that this service is dependent upon, and provides a 'tag' name to identify the service when referencing it within this service. *List.*
Messages *+xns/Discovery/MessageDefList*	The messages defined by the service. The message definition allows the caller to determine how to use the message. *List.*

Service: Folder

+xns/Folder

Enables the identity owner to organize attributes and attribute groups the same way files are organized in a directory tree. With the folder service, the identity owner can create, delete, or rename folders and subfolders. In addition, the identity owner can add data to a folder, remove data from a folder, or rename the data within a folder.

Message Summary

AddItem	Adds a data item to the folder with the specified name.
Create	Creates a new subfolder at the specified path.
Delete	Deletes a folder at the specified path.
RemoveItem	Removes the reference to a specified data item from the folder.
Rename	Relocates a folder to a new path under a new name.
RenameItem	Replaces the name of an item in a folder with a new name.

Service Dependencies

+xns/Core

Message: AddItem

+xns/Folder/AddItem
Derived from +xns/Core/Message

Adds a data item to the folder with the specified name.

The data item is added to the folder by reference (i.e., the folder does not contain the actual data item).

- **To:** Identity containing the folder to which the item will be added.
- **From:** Identity or exposure requesting the addition.

Input

Item *+xns/Core/XNSAddress*	The address of the data item to be added. *Required.*
ItemName String	The name of the item to be added. *Required.*
Path *+xns/Core/XNSName*	The address of the folder to which the item is to be added. *Required.*

Exceptions

InvalidAddress	Invalid address specified.

Message: Create

+xns/Folder/Create
Derived from +xns/Core/Message

Creates a new subfolder at the specified path.

- **To:** Identity to contain the subfolder.
- **From:** Identity or exposure requesting the creation of the subfolder.

Input

Name String	Name of the subfolder to be created.
Path *+xns/Core/XNSName*	Address where the new subfolder is to be created.

Exceptions

PathNotFound	The specified path was not found.

Message: Delete

+xns/Folder/Delete
Derived from +xns/Core/Message

Deletes a folder at the specified path.

If the folder contains subfolders, the subfolders will also be deleted. If the folder contains data references, the references will be deleted, but the data instances will remain in the identity.

- **To:** Identity that contains the folder.
- **From:** Identity or exposure requesting the deletion.

Input

Path *+xns/Core/XNSName*	Address of the folder to be deleted.

Exceptions

PathNotFound	The specified path was not found.

Message: RemoveItem

+xns/Folder/RemoveItem
Derived from +xns/Core/Message

Removes the reference to a specified data item from the folder. This message removes just the reference in the folder, not the data item itself.

- **To:** Identity containing the folder from which the data item is to be removed.
- **From:** Identity or exposure requesting the removal.

Input

ItemName String	Name of the item to be removed. *Required.*
Path *+xns/Core/XNSName*	Address of the folder from which the item is to be removed. *Required.*

Exceptions

NameNotFound	The specified item name was not found.

Message: Rename

+xns/Folder/Rename
Derived from +xns/Core/Message

Relocates a folder to a new path under a new name.

The folder specified by Path will be moved to a folder specified by NewPath, with any necessary folders being created in the process.

- **To:** Identity containing the folder to be renamed.
- **From:** Identity or exposure requesting the renaming.

Input

NewPath *+xns/Core/XNSName*	The address of the new folder. *Required.*
Path *+xns/Core/XNSName*	The address of the folder to be renamed. *Required.*

Exceptions

PathExists	The specified path already exists.

Message: RenameItem

+xns/Folder/RenameItem
Derived from +xns/Core/Message

Replaces the name of an item in a folder with a new name.

- **To:** Identity containing the folder with the item to be renamed
- **From:** Identity or exposure requesting the renaming.

Input

ItemName String	Old name of the item to be renamed. *Required.*
NewName String	New name for the item being renamed. *Required.*
Path *+xns/Core/XNSName*	Address of the folder containing the item to be renamed. *Required.*

Exceptions

NameNotFound	The specified item name was not found.

Service: Hosting

+xns/Hosting

Provides the ability to create identities, delete identities, and manage a hosting association.Identities receive messages via communication channels (URIs). A hosting association is formed when an identity (the hosted identity) wants to share communication channels managed by another identity (the host identity). A host identity manages the set of communication channels for a community of identities at the same location, called a host community. An identity can either have its own communication channels (self-hosted) or share those of a host identity. If there are many identities at the same location, a host identity adds efficiency.

Message Summary

DeleteIdentity	Removes the identity document, releases all names associated with the identity, and breaks the association between the identity ID and its URIs.
HostIdentity	Establishes a relationship between an identity and the identity that will function as its host.

Data Summary

Host	An identity that represents a network endpoint.
IDSP	ID service provider; a collection of identities that share a common network endpoint.

Service Dependencies

+xns/Core
+xns/Data

Message: DeleteIdentity

+xns/Hosting/DeleteIdentity
Derived from +xns/Core/Message

Removes the identity document, releases all names associated with the identity, and breaks the association between the identity ID and its URIs. (See *+xns/Name/Release.)*

- **To:** Host identity.
- **From:** Any identity or exposure.

Input

IdentityAddr *+xns/Core/IdentityAddress*	The address of the identity to be deleted. *Required.*

Message: HostIdentity

+xns/Hosting/HostIdentity
Derived from +xns/Core/Message

Establishes a relationship between an identity and the identity that will function as its host.

This message is sent to the identity that is to function as a host identity. A host identity shares all its URIs with the identities it hosts.

An input identity is optional. If an input identity is provided, then the input identity will be hosted by the identity receiving the message. If no input identity is provided, then a new identity is created and hosted by the identity receiving the message. The input or new identity will receive communication via the same URIs as the host identity (see *+xns/ID/Register).*

This message returns the identity address based upon the new hosting environment.

- **To:** Host identity.
- **From:** Any identity or exposure.

Input

Identity *+xns/Data/Identity*	The data of the identity that is to be hosted.

Output

IdentityAddr *+xns/Core/IdentityAddress*	The identity's new address with its host.

Data: Host

+xns/Hosting/Host
Derived from +xns/Core/GroupOwner

An identity that represents a network endpoint. The host is the owner of an ID service provider group.

Data Elements

IDSP *+xns/Hosting/IDSP*	The ID service provider group owned by this host identity.

Data: IDSP

+xns/Hosting/IDSP
Derived from +xns/Core/Group

ID service provider; a collection of identities that share a common network endpoint. The IDSP group owner is a host identity.

Data Elements

Host *+xns/Hosting/Host*	The identity that owns the ID service provider group.

Service: ID

+xns/ID

Creates and manages persistent, unchanging XNS IDs. IDs can be created and permanently retired. (Once assigned, an ID is never re-issued.) IDs are associated with communication channels (URIs) through a host identity. If a host identity is relocated on the network, its ID can be associated with different URIs. If a hosted identity moves, its ID can be associated with different hosts. Because links between identities are based on XNS IDs and not URIs, the ID service provides identities with the ability to move to a new hosting environment without breaking the links. The ID service is provided only by host identities.

Message Summary

MapID	Maps external identifiers to internal identifiers.
Register	Associates an identity ID with locations, or changes registration information if the ID is already registered.
Resolve	Resolves an identity ID to the URIs used to channel communication to the identity.
Retire	Permanently retires the identity ID.

Data Summary

CachedID	An ID that has been registered by another host identity, which this host identity is keeping for performance reasons.
ID	The registration of an XNS identity.
IDMap	Map of external values to internal instances or identities.

Datatype Summary

Map	External to internal identifier map.

Service Dependencies

+xns/Core

Message: MapID

+xns/ID/MapID
Derived from +xns/Core/Message

Maps external identifiers to internal identifiers. This message can act as a map query as well as an add, update, or delete of the map. If the input map has external but no internal identifiers, then the message returns the map with the internal identifiers filled in (map query). If the input Map parameter has both external and internal identifiers, the map managed by the identity will be updated with the contents of the input map (add or update). If the internal identifer is a single dot (“.”), the host will remove the existing map entry for the corresponding external identifier (delete).

- **To:** Any identity.
- **From:** Usually an application authenticated as the to-identity.

Input

Context String	ID context in which the external IDs included in the map are defined. For example, a host identity ID is a context for mapping external “identity IDs” to internal identity IDs; an identity ID is a context for mapping external “data IDs” to internal data IDs; “MSPassport” could be the context for mapping passport IDs to XNS IDs without requiring a host ID be assigned to Passport.
Map *+xns/ID/Map*	Mapping of external to internal identifiers. Mapping of external to internal identifiers. As an input parameter, it contains the map entries to select, add, update, or delete.

Output

Map *+xns/ID/Map*	Mapping of external to internal identifiers. Mapping of external to internal identifiers. As an output parameter, it contains the mappings requested on the input (if any).

Message: Register

+xns/ID/Register
Derived from +xns/Core/Message

Associates an identity ID with locations, or changes registration information if the ID is already registered.

An input identity ID is optional. If no identity ID is provided, then one is assigned and returned.

Either locations or a host identifier must be input. If locations are provided, the identity ID is associated with those locations. If a host identifier is provided, the identity ID is associated with the host identity and, transitively, with all the locations contained within the host identity. If the identity ID is already associated with a host identity, then the existing association is replaced with one to the specified host identity.

While an identity ID can be associated with only one host identity, the host identity may be directly associated with multiple URIs.

- **To:** A host identity.
- **From:** Another host identity or an application acting on behalf of the to-identity.

Input

HostID *+xns/Core/HostID*	The ID of the host which will be associated with the identity being registered.
IdentityID *+xns/Core/IdentityID*	The identity ID of the identity being registered. If IdentityID is not provided as an input parameter, then it is assigned and returned.
IsHost Boolean	Is the requested ID for a host identity? True=the requested ID is for a host identity; False=the requested ID is not for a host identity.
Locations *+xns/Core/TextList*	The URIs which will be associated with the identity being registered. This element applies only to host identities. *List.*

Output

IdentityID *+xns/Core/IdentityID*	The identity ID of the identity being registered. If IdentityID is not provided as an input parameter, then it is assigned and returned.

Exceptions

HostNotFound	Host not found.

Message: Resolve

+xns/ID/Resolve
Derived from +xns/Core/Message

Resolves an identity ID to the URIs used to channel communication to the identity.

If the identity's URIs are managed by a host identity, then the host identity ID is returned along with the URIs.

- **To:** Any host identity.
- **From:** Any identity or an application acting on behalf of the to-identity.

Input

IdentityID *+xns/Core/IdentityID*	The ID path of the identity to be resolved. *Required.*
NoCache Boolean	Is a cached value unacceptable? True=do not accept a cached value—return only a value retrieved from the definition source; False=accept a cached value.

Output

Cached Boolean	Was the result determined from cached data? True=the result was determined based on a cached value; False=the result was received from the original registrar within a reasonable timeframe (usually 1 hour). The timeframe allows for immediate self-healing of changed URIs, but prevents repeated requests from one host to an ID service if the resolved host URIs are down for maintenance.
HostID *+xns/Core/HostID*	The ID of the located identity's host.
IdentityID *+xns/Core/IdentityID*	The ID path of the identity to be resolved.
Locations *+xns/Core/TextList*	The URIs of the resolved identity's host. *List.*

Exceptions

IdentityNotFound	"HostNotFound" "IdentityRetired"

Message: Retire

+xns/ID/Retire
Derived from +xns/Core/Message

Permanently retires the identity ID. A locate request for this identity will return the 'IdentityRetired' exception rather than the 'NotFound' exception.

- **To:** The host identity holding the ID object for the specified identity ID.
- **From:** The host identity of the identity being retired or an application acting on behalf of the to-identity.

Input

IdentityID *+xns/Core/IdentityID*	The ID of the identity to be retired. *Required.*

Exceptions

IdentityIDNotFound	The requested identity was not found.

Data: CachedID

+xns/ID/CachedID
Derived from +xns/ID/ID

An ID that has been registered by another host identity, which this host identity is keeping for performance reasons. CachedID objects are not definitive because the identities they represent may have moved.

Data: ID

+xns/ID/ID
Derived from +xns/Core/XNSObject

The registration of an XNS identity. The ID object is held by a host identity. The Name attribute (inherited from XNSObject) of the independent ID object must contain the same contents as the RegisteredID element.

Data Elements

HostID *+xns/Core/HostID*	The XNS ID of this identity's current host identity. For host identities, the contents of this element are the same as the contents of RegisteredID.
Locations *+xns/Core/TextList*	The URIs of a host identity. Each URI describes a different connector available for communicating with the host identity and its hosted identities. This element applies only to host identities. *List.*
RegisteredID String	The ID that this object represents.
Retired **Boolean**	Is this ID retired? True=identity ID has been retired through the *+xns/ID/Retire* message; False=identity ID is active and can be communicated with.

Data: IDMap

+xns/ID/IDMap
Derived from +xns/Core/XNSObject

Map of external values to internal instances or identities

Data Elements

Map *+xns/ID/Map*	Map of external values to internal instances or identities. A map may be used to equate two distributed instances of a shared piece of data. A map may also be used to provide an internal identifier for an external identity ID.

Datatype: Map

External to internal identifier map. The map contains individual map entries consisting of a translation between an external identifier and an internal identifier. External identifiers are usually defined as "how someone else identifies the object," and internal identifiers are usually defined as "how I identify the object."

```
EBNF:Map ::= MapEntry * (' ' MapEntry)
MapEntry ::= [ExternalIDPath |  ExternalNamePath] '!' InternalIDPath
ExternalIDPath ::= IDPath
ExternalNamePath ::= NamePath
InternalIDPath ::= IDPath
IDPath ::= ID * ('.' ID)
NamePath ::= Name * ('/' Name)
ID ::= XML text string normalized according to the XNS ID normalization
  rules
Name ::= XML text string normalized according to the XNS name
  normalization rules
```

Service: Name

+xns/Name

Creates and manages unique semantic names within a given name space. A name can be associated with an existing ID, released for use by another identity, replaced with a different name, or reserved while the identity owner completes the name registration process. Some objects in the Name service are designed to be held by the registrar, while other objects are designed to be held by the registrant.

Message Summary

CloseNamespace	Closes an existing namespace for further registrations.
Extend	Extends the registration period for an XNS name.
GetValidNamespaces	Returns a list of namespaces the identity has opened for registration.
OpenNamespace	Declares a namespace for holding name registrations.
Register	Associates a unique XNS name with an established identity address.
Release	Breaks the association between an identity ID and an XNS name and renders that name available for use by another identity.
Rename	Registers a new XNS name and releases an old XNS name as a single operation.
Request	Checks to see if an XNS name is available and, optionally, places a reservation on that name.
Resolve	Resolves an identity name to an identity address.
ResolveNamespace	Determines the identity that owns the specified namespace.
Transfer	Transfers the registration of an XNS name from one identity address to another.

Data Summary

CachedName	A name that has been registered by another name service, which this name service is keeping for performance reasons.
MyName	The registration of an XNS name held by the identity that the name represents.
Name	The registration of an XNS name to an XNS address held by the name registrar.
ServiceProfile	Identity-specific preferences for the Name service.

Service Dependencies

+xns/Core

Message: CloseNamespace

+xns/Name/CloseNamespace
Derived from +xns/Core/Message

Closes an existing namespace for further registrations. Names within this namespace continue to be valid, but the +xns/Name/ Register message will not allow new names to be registered under this namespace.

- **To:** Identity owning the namespace to be closed.
- **From:** Usually, an application running on behalf of a person authenticated as the owner of the to-identity.

Input

Namespace *+xns/Core/XNSName*	Namespace to close for further name registrations.

Exceptions

NotOpen	The specified namespace is not an existing namespace open for registrations.

Message: Extend

+xns/Name/Extend
Derived from +xns/Core/Message

Extends the registration period for an XNS name.

- **To:** The registrar identity for this name.
- **From:** Usually the identity the name is being extended for, but can be any identity the registrar grants extension authority.

Input

ExpirationDate *+xns/Core/XMLDateTime*	New expiration date for this XNS name.
Name *+xns/Core/XNSName*	XNS name for which the expiration date is to be extended.

Message: GetValidNamespaces

+xns/Name/GetValidNamespaces
Derived from +xns/Core/Message

Returns a list of namespaces the identity has opened for registration.

- **To:** Any identity.
- **From:** Any identity or application.

Output

Namespaces *+xns/Core/XNSNameList*	The namespaces the identity has opened for registration.*List.*

Message: OpenNamespace

+xns/Name/OpenNamespace
Derived from +xns/Core/Message

Declares a namespace for holding name registrations. Names cannot be registered into a namespace (using *+xns/Name/Register)* until that namespace has been opened.

- **To:** Identity being asked to open a namespace.
- **From:** Usually, an application running on behalf of a person authenticated as the owner of the to-identity.

Input

Namespace *+xns/Core/XNSName*	Space within the identity to hold name registrations.

Exceptions

NameTaken	The specified name is already taken and cannot be used as a namespace.

Message: Register

+xns/Name/Register
Derived from +xns/Core/Message

Associates a unique XNS name with an established identity address.

- **To:** Identity functioning as a name registrar.
- **From:** Any identity or exposure.

Input

IdentityAddress *+xns/Core/IdentityAddress*	Identity address to which the new XNS name will be registered. If no address is provided, the address of the message from- identity is used.
Name *+xns/Core/XNSName*	Unique name that conforms to the XNS name specification.
ReservationID String	Identifier of a reservation placed against this name by the *+xns/Name/Request* message.

Output

ExpirationDate *+xns/Core/XMLDateTime*	The date upon which this name must be re-registered in order to be retained.
FullName *+xns/Core/XNSName*	Fully qualified version of the newly registered name. For example, if an identity sends a *+xns/Name/Register* message with Name set to "foo" to an identity with a PreferredName of @mycompany, the fully qualified name returned would be @mycompany/foo. On the other hand, if an identity sends a *+xns/Name/Register* message with name set to @myowncompany/foo to an identity with a preferred name of @mycompany but an additional registered name of @myowncompnay, the explicit preference of the caller would be respected and the fully qualified name returned would be @myowncompany/foo.

Message: Release

+xns/Name/Release
Derived from +xns/Core/Message

Breaks the association between an identity ID and an XNS name and renders that name available for use by another identity.

- **To:** Identity functioning as a name registrar.
- **From:** Any identity or exposure.

Input

Name *+xns/Core/XNSName*	The XNS name to be released.

Exceptions

NameNotFound	Name requested to be released was not found.

Message: Rename

+xns/Name/Rename
Derived from +xns/Core/Message

Registers a new XNS name and releases an old XNS name as a single operation. If there are any problems registering the new name, then the old name is not released.

- **To:** Identity functioning as a name registrar.
- **From:** Any identity or exposure.

Input

NewName *+xns/Core/XNSName*	The new XNS name which will replace the old one. *Required.*
OldName *+xns/Core/XNSName*	The XNS name to be replaced. *Required.*
ReservationID String	Identifier of a reservation placed against the new name by the *+xns/Name/Request* message.

Message: Request

+xns/Name/Request
Derived from +xns/Core/Message

Checks to see if an XNS name is available and, optionally, places a reservation on that name.

- **To:** Identity functioning as a name registrar.
- **From:** Any identity or exposure.

Input

Name *+xns/Core/XNSName*	The XNS name to be checked for availability. *Required.*
Reserve Boolean	Should a reservation be placed against the requested name if it is available? True=reserve requested name; False=do not place a reservation on the requested name.

Output

ReservationID String	Identifier of a reservation placed against the requested name on behalf of the requesting identity.

Exceptions

NameNotAvailable	The requested name is already assigned to another identity.

Message: Resolve

+xns/Name/Resolve
Derived from +xns/Core/Message

Resolves an identity name to an identity address.

- **To:** Identity who provides the Name service.
- **From:** Any identity or exposure.

Input

Name *+xns/Core/XNSName*	The name of the identity to be resolved. Name is not utilized by the *+xns/Name/ResolveNamespace* message, which extends Resolve. *Required.*
NoCache Boolean	Is a cached value unacceptable? True=do not accept a cached value—return only a value retrieved from the definition source; False=accept a cached value.
PartialResolve Boolean	Is partial resolution acceptable? True=resolve as much as can be resolved locally, returning enough information to know where to go next; False=forward the call until resolution is complete, returning the fully resolved name.

Output

Cached Boolean	Was the result determined from cached data? True=the result was determined based on a cached value; False=the result was determined from the original registrar and is definitive.
ExpirationDate *+xns/Core/XMLDateTime*	The date upon which this name must be re-registered in order to be retained.
IdentityAddress *+xns/Core/IdentityAddress*	The resolved address of the identity that owns the name or namespace.
PartiallyResolved Boolean	Is this a partially resolved value? True=the PartialResolve input argument was true and the resolved identity is the identity that owns the namespace contained in RelativeRemainder; False=resolution is complete.
RelativeRemainder *+xns/Core/XNSName*	Name remaining after partial resolution, relative to the partially resolved identity.

Exceptions

NameNotFound	Name was not found.

Message: ResolveNamespace

+xns/Name/ResolveNamespace
Derived from +xns/Name/Resolve

Determines the identity that owns the specified namespace. This message clarifies if a given namespace refers to an identity or a namespace within an identity. For example, given the Namespace "@mycompany/Region/Northeast," it is unknown if Northeast is an identity or a namespace within an identity. It is also unclear if Region is an identity or a namespace within the @mycompany identity. If @mycompany owns "/Region/Northeast," then the message will return the IdentityAddress of @mycompany and the RelativeRemainder of "/Region/Northeast." If Region is an identity managing the "/Northeast" namespace, then the message will return the IdentityAddress of the @mycompany/Region identity, and the "/Northeast" as the RelativeRemainder.

If the specified namespace begins with "/xns" (or any of the shortcuts +, =, or @), then the message can be sent to any identity. If the specified namespace does not begin with "/xns," then the message must be sent to the identity that owns the first name of the specified namespace.

- **To:** Any identity if Namespace begins with "/xns"; otherwise, identity owning the top of the specified namespace.
- **From:** Any identity or application.

Input

Namespace *+xns/Core/XNSName*	Namespace for which to determine the owner.

Message: Transfer

+xns/Name/Transfer
Derived from +xns/Core/Message

Transfers the registration of an XNS name from one identity address to another.

- **To:** The registrar identity for this name.
- **From:** Usually the identity the name is being transferred from, but can be any identity the registrar grants transfer authority.

Input

IdentityAddress *+xns/Core/IdentityAddress*	Identity address to which the XNS name will be transferred.
Name *+xns/Core/XNSName*	XNS name to be transferred.

Data: CachedName

+xns/Name/CachedName
Derived from +xns/Name/Name

A name that has been registered by another name service, which this name service is keeping for performance reasons. *(+xns/Name/Name* is a name that this name service has registered.) CachedName objects may be removed, but can always be resolved again.

Data: MyName

+xns/Name/MyName
Derived from +xns/Core/XNSObject

The registration of an XNS name held by the identity that the name represents.

Data Elements

ExpirationDate *+xns/Core/XMLDateTime*	The date upon which this name must be re-registered in order to be retained.
RegisteredName *+xns/Core/XNSName*	The registered name.
Registrar *+xns/Core/IdentityAddress*	The address of the registrar identity for this name.
ReservationID String	Identifier of a reservation placed against this name by the requesting identity.

Data: Name

+xns/Name/Name
Derived from +xns/Core/XNSObject

The registration of an XNS name to an XNS address held by the name registrar. The Name attribute (inherited from XNSObject) of the independent Name object must contain the contents of RegisteredName.

Data Elements

Address *+xns/Core/IdentityAddress*	The address of the identity to whom this name is registered.
ExpirationDate *+xns/Core/XMLDateTime*	The date upon which this name must be re-registered in order to be retained.
RegisteredName *+xns/Core/XNSName*	The registered name.
ReservationID String	Identifier of a reservation placed against this name by the requesting identity.

Data: ServiceProfile

+xns/Name/ServiceProfile
Derived from +xns/Core/XNSObject

Identity-specific preferences for the Name service.

Data Elements

PreferredName *+xns/Name/MyName*	The registered name that this identity prefers to be known as. Whan an identity publishes its XNS address, this will be the XNSName on the right hand side of the address. If null, there will be no XNSName component of the identity's XNSAddress.
ValidNamespaces *+xns/Core/XNSNameList*	The namespaces the +xns/Name/Register message may register names within. *List.*

Service: Negotiation

+xns/Negotiation

Manages links and contracts between identities that govern the exchange of data and access to services. An identity link can specify what data is to be exchanged with another identity, as well as the data protection controls and permissions that apply to that data and if and how the data is to be kept synchronized. Likewise, an identity can specify what services it will provide to another identity and the terms under which those services will be provided. In addition, the terms of an existing contract can be renegotiated or the contract terminated.

Message Summary

ConfirmNegotiation	Declares that a pending contract has either been accepted or rejected.
ConfirmReceiptAccepted	Notifies the originating identity that the receipt has been accepted.
NegotiateContract	Arrives at an agreement between two identities.
RequestForm	Requests a contract form from an identity.
SubmitForConfirmation	Allows the non-originating identity to provide data for the contract.
SubmitForNegotiation	Prepares the contract for negotiation.
SubmitReceipt	Submits the contained receipt for acceptance.
TerminateContract	Ends an existing contract.

Data Summary

Contract	An agreement between two identities governing data access, data usage, and service entitlements.
ContractDataSet	Container for the data governed by the contract.
ContractTerms	A collection of policies that inform a contract, including security and privacy policies.
ContractTypeEnum	Valid data exchange types that a contract can specify.
DataAccessPermission	A permission granted for a set of data within a contract specifying what actions can be performed against the data.
DataDistribution	A group of data an identity is authorized to distribute.
DataPermission	A permission granted for a set of data within a contract.

Data Summary

DataPermissionRequest	A request for a permission around how a set of data within a form can be accessed or used.
DataRequest	Data object which is requested on a form.
Form	A template for a contract.
Link	A relationship with another identity.
MessagePermission	Permission to invoke a particular message.
PermissionTypeEnum	Uses for which permission may be granted.
PrivacyPermission	A permission granted for a set of data within a contract specifying how the receiving identity can use the data supplied by the providing identity.
PrivacyPermissionRequest	A request for a permission around how a receiving identity can use, retain, and disclose data supplied by a providing identity.
Receipt	Record of a product or service performed.
ReceiptDetail	A set of supporting information for a receipt.
SecurityLevelEnum	Valid transport layer security requirement levels.
SyncLevelEnum	Valid data synchronization levels.
SyncPermission	A permission granted for a set of data within a contract specifying how data provided by an identity is updated to the receiving identity.
SyncPermissionRequest	A request for a permission around how a receiving identity gains access to data supplied by a providing identity.

Service Dependencies

+xns/Certification
+xns/Core

Message: ConfirmNegotiation

+xns/Negotiation/ConfirmNegotiation
Derived from +xns/Core/Message

Declares that a pending contract has either been accepted or rejected.

This message is sent to an identity who has previously requested a contract and been told that the contract status is pending, i.e., the original *+xns/Negotiation/NegotiateContract* message was returned with the Negotiated flag set to False. (A pending contract usually means the identity needs to get manual acceptance from the principal.) Upon completion of negotiation, this message is sent to the requesting identity as a notification that the contract was either accepted or rejected.

- **To:** Identity who originated contract negotiations.
- **From:** Identity originally asked to enter into the contract.

Input

Accepted Boolean	Has the contract been accepted? True=the contract has been accepted; False=the contract was rejected. *Required.*
Contract *+xns/Negotiation/Contract*	Contract with data provided.
ContractName String	The name of the contract that has been negotiated. *Required.*

Exceptions

UnknownName	A contract by this name cannot be found.

Message: ConfirmReceiptAccepted

+xns/Negotiation/ConfirmReceiptAccepted
Derived from +xns/Core/Message

Notifies the originating identity that the receipt has been accepted. A receipt submitted earlier via a *+xns/Negotiation/SubmitReceipt* has now been either accepted or rejected, and this is the notification thereof.

- **To:** Identity originating the receipt.
- **From:** Identity receiving the receipt.

Input

Accepted Boolean	Was the receipt accepted? True=the principal has accepted the receipt; False=the principal has rejected the receipt, and may have provided the reason in the ReasonForRejection element.
ReasonForRejection String	Textual description for the reason the receipt was rejected. If the receipt was rejected (Accepted=False), the principal may provide the reason in this element.
Reference String	Receipt identification. This is the reference number that identifies the receipt being accepted. *Required.*

Message: NegotiateContract

+xns/Negotiation/NegotiateContract
Derived from +xns/Core/Message

Arrives at an agreement between two identities.

- **To:** Identity being asked to negotiate the contract; usually the creator of the form for the contract.
- **From:** Identity requesting the negotiation; usually the data or service provider.

Input

Contract *+xns/Negotiation/Contract*	The contract document under negotiation. *Required.*

Output

Negotiated Boolean	Is the contract negotiation complete? True=negotiation complete; False=contract pending manual negotiation.

Exceptions

NoAgreement	A contract term was specified which could not be agreed upon.

Message: RequestForm

+xns/Negotiation/RequestForm
Derived from +xns/Core/Message

Requests a contract form from an identity. A form is a template for a contract. It describes the data being requested and the terms of use for that data. The requestor creates a form, the data provider gets the form (using this message), and decides if he wants to provide the data to the requestor under these terms.

- **To:** Identity who owns the form.
- **From:** Identity who wants to negotiate a contract; usually the data or service provider.

Input

FormAddr *+xns/Core/XNSAddress*	The address of the form being requested. *Required.*

Output

Form *+xns/Negotiation/Form*	The requested form.

Exceptions

AccessDenied	The form could not be accessed.

Message: SubmitForConfirmation

+xns/Negotiation/SubmitForConfirmation
Derived from +xns/Core/Message

Allows the non-originating identity to provide data for the contract. When the originating identity is not the data provider, a contract will be left in pending status on both sides until the non- originating identity fills the contract with data. When that happens, this message is called to send the data to the originator. This message calls *+xns/Negotiation/ConfirmNegotiation* to push the filled-in data to the originator.

- **To:** Non-originating identity of the contract.
- **From:** Non-originating identity of the contract.

Input

Accepted Boolean	Has the contract been accepted? True=the contract has been accepted; False=the contract was rejected. *Required.*
Contract *+xns/Negotiation/Contract*	Contract with data provided.
ContractName String	Contract name for which to add data. *Required.*

Exceptions

UnknownName	A contract by this name cannot be found.

Message: SubmitForNegotiation

+xns/Negotiation/SubmitForNegotiation
Derived from +xns/Core/Message

Prepares the contract for negotiation. After retrieving a form, the contract originator decides if he wants to provide data to the requestor under the terms of the form. If so, the originator builds a contract (using this message). This message completes the contract by setting the date and transferring required data from the form. The contract is then sent to the data requestor using the *+xns/Negotiation/ NegotiateContract* message.

- **To:** Identity originating the contract.
- **From:** Identity originating the contract.

Input

Contract *+xns/Negotiation/Contract*	Proposed contract, partially filled in. *Required.*
Form *+xns/Negotiation/Form*	Form used as basis for the contract. *Required.*
NegotiateWith *+xns/Core/IdentityAddress*	Address of the identity to negotiate with contract with. *Required.*

Output

Negotiated Boolean	Is the contract negotiation complete? True=negotiation is complete; False=contract is pending manual negotiation.

Message: SubmitReceipt

+xns/Negotiation/SubmitReceipt
Derived from +xns/Core/Message

Submits the contained receipt for acceptance.

- **To:** Identity receiving the receipt.
- **From:** Identity originating the receipt.

Input

OwnerRequired Boolean	Is the principal required to accept the receipt? True=the principal must accept the receipt; False=the identity may accept the receipt if the principal has empowered it to do so.
Receipt *+xns/Negotiation/Receipt*	The receipt being submitted. *Required.*

Output

Accepted Boolean	Was the receipt accepted? True=the identity was empowered to accept this receipt and has accepted it; False=the receipt is pending acceptance by the principle and notification will follow in the form of a *+xns/Negotiation/ConfirmReceiptAccepted* message.

Exceptions

NotAccepted	The receipt was rejected. A reason for rejection may be included in the exception text.

Message: TerminateContract

+xns/Negotiation/TerminateContract
Derived from +xns/Core/Message

Ends an existing contract. This message is sent by one identity involved in a contract to the other identity involved in the contract, specifying the contract to be terminated. Ether identity can orginate this message.

- **To:** Identity being asked to terminate the contract.
- **From:** Identity requesting contract termination.

Input

ContractName String	The name of the contract to be terminated. *Required.*

Exceptions

TerminationProhibited	The contract may specify terms that prohibit termination under certain conditions, such as an agreed upon termination date.

Data: Contract

+xns/Negotiation/Contract
Derived from +xns/Core/XNSObject

An agreement between two identities governing data access, data usage, and service entitlements. Each identity stores a copy of the contract. The Name attribute (inherited from XNSObject) of the independent contract object must contain the contents of ContractID.

Data Elements

BeginDate *+xns/Core/XMLDateTime*	Effective date of the contract.
ContractDate *+xns/Core/XMLDateTime*	Date the contract negotiation was completed.
ContractID String	An identifier generated by the originator and used by messages to uniquely identify the contract.
ContractTerms *+xns/Negotiation/ ContractTerms*	Terms and conditions governing the contract.
ContractType *+xns/Negotiation/ ContractTypeEnum*	Identifier of the contract as an extension, subscription, distribution, or service contract.
DataAccessPermissions *+xns/Negotiation/ DataAccessPermissionList*	List of data access permissions agreed upon.
DataSet *+xns/Negotiation/ ContractDataSetList*	Container for the sets of data governed by the contract.
EndDate *+xns/Core/XMLDateTime*	Date the contract is due to expire.
Form *+xns/Core/XNSAddress*	Address of the form used as the template for this contract. This address must be a versioned address so that the identity entering the contract can always retrieve the form definition.
GroupID *+xns/Core/DataID*	Address of the group whose members participate in this contract. If this is null, the contract does not apply to a group, but to the identity who owns this contract only.
MessagePermissions *+xns/Negotiation/ MessagePermissionList*	The message permissions agreed upon. Message permissions are provided only for service contracts. *List.*

Data Elements

Originator *+xns/Core/IdentityAddress*	Address of the identity originating the contract. This is the identity who creates the contract and sends it to the other identity in the *+xns/Negotiation/NegotiateContract* message.
Pending Boolean	Is this contract pending negotiation? If the identity(s) are not empowered to negotiate the contract, the contract will be set to Pending, awaiting manual approval. Once approved (via a *+xns/Negotiation/ApproveContract* message), the Pending flag will be turned off, and a *+xns/Negotiation/ConfirmNegotiation* message sent to the originator.
PrivacyPermissions *+xns/Negotiation/PrivacyPermissionList*	Data privacy permissions. Because privacy permissions apply to data, privacy permissions are not provided for service contracts. *List.*
Provider *+xns/Core/IdentityAddress*	Address of the identity providing the data or service.
SyncPermissions *+xns/Negotiation/SyncPermissionList*	Data synchronization permissions. Because synchronization applies to data, synchronization permissions are not provided for service contracts. *List.*
TerminatedBy *+xns/Core/IdentityID*	Address of the identity that terminated the contract.
TerminationDate *+xns/Core/XMLDateTime*	The date on which the contract was terminated. This will be set only on terminated (vs. expired) contracts.

Data: ContractDataSet

+xns/Negotiation/ContractDataSet
Derived from +xns/Core/XNSObject

Container for the data governed by the contract.

Data Elements

Data *+xns/Core/XNSObjectList*	The data contained in this contract data set. The name of each XNS object in the list must match the Name attribute (inherited from XNSObject) in the associated data request object in the form definition. List.

Data: ContractTerms

+xns/Negotiation/ContractTerms
Derived from +xns/Core/XNSObject

A collection of policies that inform a contract, including security and privacy policies.

Data Elements

HTMLPrivacyPolicy String	The URI of the data or service receiver's human readable privacy policy.
HTMLSecurityPolicy String	The URI of the data or service receiver's human readable security policy.
HTMLTermsPolicy String	The URI of the data or service receiver's human readable terms and conditions policy.
P3PPrivacyPolicy String	The URI of the data or service receiver's P3P privacy policy.
PurposeText String	Description of the receiver's intended use of the data or service.

Data: ContractTypeEnum

+xns/Negotiation/ContractTypeEnum
Derived from +xns/Core/XNSObject

Valid data exchange types that a contract can specify. The enumeration values include:

- **EXTENSION**—Governs the distribution of data to third parties.
- **SUBSCRIPTION**—Governs access by an identity to data owned by another identity.
- **SERVICE**—Governs performance of certain actions.

Data: DataAccessPermission

+xns/Negotiation/DataAccessPermission
Derived from +xns/Core/XNSObject

A permission granted for a set of data within a contract specifying what actions can be performed against the data.

Element Details

CanCreate Boolean	Can an instance of the data be created? True=an instance of the data may be created; False=permission is not granted to create an instance of the data.
CanDelete Boolean	Can an instance of the data be deleted? True=an instance of the data may be deleted; False=permission is not granted to delete an instance of the data.
CanRead Boolean	Can the data be read? True=the data may be read; False=permission is not granted to read the data.
CanUpdate Boolean	Can an instance of the data be updated? True=an instance of the data may be updated; False=permission is not granted to update an instance of the data.
DataIDs *+xns/Core/DataIDList*	Addresses of the data that this permission covers. All data under these nodes are also covered by this permission. *Required.*

Data: DataDistribution

+xns/Negotiation/DataDistribution
Derived from +xns/Core/XNSObject

A group of data an identity is authorized to distribute. Objects represented in this distribution are usually not owned by the owner of this distribution object, but given to the owner under XNS contract for the purposes of distribution.

Data Elements

Data *+xns/Core/ElementList*	Elements of data contained in this distribution. *List.*
DistributeTo *+xns/Core/IdentityDataAddressList*	Addresses of XNS data schema definitions defining the kinds of XNS identities to which data is to be distributed. *List.*
SchemaAddress *+xns/Core/IdentityDataAddress*	Address of the schema defining the type of objects contained in this distribution.

Data: DataPermission

+xns/Negotiation/DataPermission
Derived from +xns/Core/XNSObject

A permission granted for a set of data within a contract.

Data Elements

DataNames *+xns/Core/TextList*	The data names to which this permission applies. Each name in this list must match a name contained in the Data element in the Contract-DataSet. *List.*
Purpose String	Human readable description of the purpose for the permission.

Data: DataPermissionRequest

+xns/Negotiation/DataPermissionRequest
Derived from +xns/Core/XNSObject

A request for a permission around how a set of data within a form can be accessed or used.

Data Elements

DataRequests *+xns/Core/TextList*	Subset of all the data requests on a form to which this data permission request applies. Each name in this list must match a Name contained in a DataRequest object of the form. *List.*
OptOut Boolean	Is this an opt-out permission? True=receiver specifies that the provider may implicitly grant this permission (i.e., permission automatically granted unless the provider specifies otherwise); False=receiver specifies that the provider must explicitly grant this permission.
Purpose String	Human readable description of the purpose for the permission request.
Required Boolean	Is this permission required? True=this permission is required; False=this permission is not required.

Data: DataRequest

+xns/Negotiation/DataRequest
Derived from +xns/Core/XNSObject

Data object which is requested on a form. The Name attribute (inherited from *+xns/Core/XNSObject)* of the independent data request object must contain the contents of RequestorName.

Data Elements

DataType *+xns/Core/IdentityDataAddress*	Address of a DataTypeDef object that defines the data type being requested.
RequestorName String	The receiver's name for this data object. Synchronization permission requests and privacy permission requests make reference to the data based on this name, and this name is used in the contract.
Required Boolean	Is this data object required? True=this data object is required; False=the data object is not required.

Data: Form

+xns/Negotiation/Form
Derived from +xns/Core/XNSObject

A template for a contract. A form includes either data or messages being requested. It also specifies the message and data permission requests.

Data Elements

ContractTerms *+xns/Negotiation/ContractTerms*	Proposed terms and conditions for governing the contract.
ContractType *+xns/Negotiation/ContractTypeEnum*	Identifier of the contract that will be created based on this form as an extension, subscription, distribution, or service contract. Contracts created based on this form will carry this value in their ContractType attribute.
DataAccessPermissions *+xns/Negotiation/DataAccessPermissionList*	Data access permissions that may be agreed to and referenced by a contract. *List.*
DataRequests *+xns/Negotiation/DataRequestList*	Data objects being requested on this form. *List.*
IdentityNegotiate Boolean	Is the identity who receives the contract resulting from this form (via the *+xns/Negotiation/NegotiateContract* message) empowered to negotiate the contract? True=identity will ensure the privacy proposals match the privacy terms in the contract and that all required data is present; False=the identity will create a PendingContractNotice object for the user to approve or reject.
MessagePermissions *+xns/Negotiation/MessagePermissionList*	Available message permissions that may be agreed to and referenced by a contract. *List.*
PrivacyRequests *+xns/Negotiation/PrivacyPermissionRequestList*	Data privacy permissions being requested on this form. *List.*
SyncRequests *+xns/Negotiation/SyncPermissionRequestList*	Data synchronization permissions being requested on this form. *List.*

Data: Link

+xns/Negotiation/Link
Derived from +xns/Core/XNSObject

A relationship with another identity. A link contains XNS contracts and other information that supports the relationship. The Name attribute (inherited from *+xns/Core/XNSObject)* of the independent Link object must contain the IdentityID of the other identity.

Contracts *+xns/Negotiation/ContractList*	List of XNS contracts with the other identity. *List.*
Identity *+xns/Core/IdentityAddress*	Address of the identity this is a link with. The IdentityID portion of this address is stored in the Name attribute of this object.
KnownAs *+xns/Core/IdentityAddress*	Identifier the other identity knows this identity as. In order to prevent triangulation based on a common identifier, it may be desirable to give out different identifiers to each identity to which this identity is linked. If a unique identifier was provided to the other identity, that unique identifier is stored in the KnownAs element.
PublicKey *+xns/Certification/PublicKeyCert*	Public key of the other identity. Public key of the other identity. Messages between identity agents are usually secured between the IDSPs hosting the identity. If it is desired to secure communications based on the identities themselves, then the public key for the other identity is stored in this element.
Receipts *+xns/Negotiation/ReceiptList*	Container for accepted and pending receipts. *List.*

Data: MessagePermission

+xns/Negotiation/MessagePermission
Derived from +xns/Core/XNSObject

Permission to invoke a particular message.

Data Elements

Condition String	Condition under which this message in the given process states may be executed. For example, for the message *+xns/Hosting/HostIdentity*, process state = "LargeIdentity", the condition may be "User agrees to pay an additional 10% hosting fee for identities over 5 megabytes."
MsgName *+xns/Core/IdentityDataAddress*	Address of a MessageDef object that defines the message to which this permission applies.
ProcessStates *+xns/Core/TextList*	States that this permission allows a message to be in. *List.*

Data: PermissionTypeEnum

+xns/Negotiation/PermissionTypeEnum
Derived from +xns/Core/XNSObject

Uses for which permission may be granted. The enumeration values include:

- **CONTACT**—Data may be used to contact the provider.
- **DISCLOSURE**—Data may be disclosed to a specified third party.
- **RETENTION**—Data may be retained by the receiver.

Data: PrivacyPermission

+xns/Negotiation/PrivacyPermission
Derived from +xns/Negotiation/DataPermission

A permission granted for a set of data within a contract specifying how the receiving identity can use the data supplied by the providing identity.

Data Elements

Agreed Boolean	Was the proposal accepted by the data provider? True=the proposal was accepted by the data provider; False=the proposal was not accepted by the data provider.
Implied Boolean	Was the permission implied, rather than being set by the privacy policy? True=the permission was not set by the data provider, but by implying the value based on the receiver's privacy policy; False=the permission was actively set by the data provider.
Party String	Person or organization to which the permission applies. f not specified, the party is the form author. For example, permission may be granted to disclose to "selected 3rd parties."
PermissionType *+xns/Negotiation/PermissionTypeEnum*	Use of the data for which permission is granted.

Data: PrivacyPermissionRequest

+xns/Negotiation/PrivacyPermissionRequest
Derived from +xns/Negotiation/DataPermissionRequest

A request for a permission around how a receiving identity can use, retain, and disclose data supplied by a providing identity. A privacy permission request is contained within a form. When the form is converted into a contract, the privacy permission request becomes a privacy permission.

Data Elements

Party String	Persons, organizations, or groups to which the permission request applies. If not specified, the party is the form author. For example, permission may be requested to disclose to "affiliates".
PermissionType *+xns/Negotiation/PermissionTypeEnum*	Use of the data for which permission is requested.

Data: Receipt

+xns/Negotiation/Receipt
Derived from +xns/Core/XNSObject

Record of a product or service performed. Receipts are created by the originating identity in a transaction and sent to the receiving identity via a *+xns/Negotiation/SubmitReceipt* message. The receipt may be accepted by the receiving identity during the SubmitReceipt message or via a *+xns/Negotiation/ConfirmReceiptAccepted* message sent at a later time.

Data Elements

Contract String	Reference to an XNS contract. If this transaction is governed by an XNS contract, this element contains the Name attribute of that contract.
Details *+xns/Negotiation/data ReceiptDetail*	Supporting information for the receipt.
Pending Boolean	Is this receipt pending acceptance from the receiver? True=the receiving identity's owner has not accepted this receipt; False=the receiving identity's owner has accepted this receipt. Pending will be set to True if the *+xns/Negotiation/SubmitReceipt* message returns a value of 'False' in the Accepted argument. When the receiving identity's owner accepts the receipt, Pending will be set to 'False' and a *+xns/Negotiation/ConfirmReceiptAccepted* message will be sent to the originating party.
Reference String	Receipt identification. This is a number or name that uniquely identifies the receipt. The originator assigns this value and defines its scope of uniqueness.
TrxDate *+xns/Core/XMLDateTime*	Transaction date. This is the date and time that the product was delivered or the service was performed.

Data: ReceiptDetail

+xns/Negotiation/ReceiptDetail
Derived from +xns/Core/XNSObject

A set of supporting information for a receipt. This is an abstract base class from which specific receipt detail classes are derived.

Data: SecurityLevelEnum

+xns/Negotiation/SecurityLevelEnum
Derived from +xns/Core/XNSObject

Valid transport layer security requirement levels. HTTPS_40 includes HTTPS_128, and NONE includes both HTTPS_40 and HTTPS_128. The enumeration values include:

- **NONE**—No security level specified.
- **HTTPS_40**—40 bit HTTPS.
- **HTTPS_128**—128 bit HTTPS.

Data: SyncLevelEnum

+xns/Negotiation/SyncLevelEnum
Derived from +xns/Core/XNSObject

Valid data synchronization levels. The value of PUSH includes ONE_TIME, and the value of PUSH_OR_PULL includes both PUSH and ONE_TIME. The enumeration values include:

- **ONE_TIME**—Single transfer of data.
- **PUSH**—Provider may push to receiver.
- **PUSH_OR_PULL**—Provider may push, and receiver may pull.

Data: SyncPermission

+xns/Negotiation/SyncPermission
Derived from +xns/Negotiation/DataPermission

A permission granted for a set of data within a contract specifying how data provided by an identity is updated to the receiving identity.

Data Elements

PushOnChange Boolean	Does the provider 'push' this data when it is changed? True=the provider pushes the data to the receiver upon update; False=the data is not pushed to the receiver.
SecurityLevel *+xns/Negotiation/SecurityLevelEnum*	Minimum transport layer security level required for synchronization.
SyncLevel *+xns/Negotiation/SyncLevelEnum*	The mode of synchronization granted by this permission. Synchronization can be one-time only, push, or both push and pull.

Data: SyncPermissionRequest

+xns/Negotiation/SyncPermissionRequest
Derived from +xns/Negotiation/DataPermissionRequest

A request for a permission around how a receiving identity gains access to data supplied by a providing identity. A synchronization permission request is contained within a form. When the form is converted into a contract, the synchronization permission request becomes a synchronization permission.

Data Elements

PushOnChangeRequired Boolean	Is the push-on-change synchronization permission required? True=the synchronization permission created in the contract from this synchronization permission request requires the PushOnChange attribute be set to True.
SecurityProposals *+xns/Core/TextList*	The security levels that this synchronization permission request will allow. When the synchronization permission request turns into a synchronization permission in the contract, the specific SecurityLevel element must contain one of these values. *List.*
SyncProposals *+xns/Core/TextList*	The synchronization levels that this synchronization permission request will allow. When the synchronization permission request turns into a synchronization permission in the contract, the specific SyncLevel element must contain one of these values. *List.*

Service: Session

+xns/Session

Supports the XNS single sign-on (SSO) application. The session service is a web browser/server-based service that allows browser-based authentication to be shared between many web sites.

Message Summary

Authenticate	Produces authentication credentials for the current browser session.
GetSessionURI	Obtains the URI for session services provided by this identity.
Login	Establishes a browser login session, then produces authentication credentials for the session.
LoginNotify	Notifies the session service of a successful login.
LogoutNotify	Notifies a merchant that the user has logged out.
SubmitAuthCert	Submits an authentication certificate to a merchant.
XNSSessionLogout	Invalidates the current authentication session.

Data Summary

ServiceProfile	Identity-specific preferences for the session service.

Service Dependencies

+xns/Core

Message: Authenticate

+xns/Session/Authenticate
Derived from +xns/Core/Message

Produces authentication credentials for the current browser session. If no session exists, the user is asked to login. The result of sending this message will be a subsequent *+xns/Session/SubmitAuthCert* message posted to the MerchantURI specified in the message. This message usually originates from the merchant, is sent to a central session service, then forwarded to the authentication authority by the session service. Upon successful authentication, an AuthCert is created for the merchant and sent to the merchant URI with the *+xns/Session/SubmitAuthCert* message.

- **To:** If specified, this is the identity to send the request to. If unspecified, authentication will be forwarded to the merchant's preferred authentication authority.
- **From:** The originator of the message. This may be the originating merchant, the central session service, or a forwarding AA.

Input

LogoutURI *+xns/Core/URI*	URI at the merchant site used to inform the merchant the user is logged out. See *+xns/Session/LogoutNotify*.
MerchantAddr *+xns/Core/IdentityAddress*	The merchant's registered XNS address. The resulting *+xns/Session/SubmitAuthCert* message will have this in its To element, and the contained AuthCert element will be encrypted using the public key of this XNS identity.
MerchantName String	Display name for the merchant originating the request. This name will be displayed to the user during logout, and should be language specific to the user (if the user's language is known).
MerchantURI *+xns/Core/URI*	URI of the merchant's session service. This URI will be used for posting the resulting *+xns/Session/SubmitAuthCert* message.
PreferredAA *+xns/Core/URI*	URI of the session service for the merchant's preferred authentication authority. If the user is not logged in, and has not specified his XNS name in the To element of the message, and has no browser plugin for capturing authentication requests, then the user will be sent to this URI for authentication. The AA will accept the user's authentication credentials, and may offer the user a chance to register at that AA if the user has no other XNS primary identity. The user may be directed to another AA if his primary XNS identity is not hosted at this AA.
SessionURI *+xns/Core/URI*	URI of the central session service used to define the browser session. The merchant sends the originating message to this URI. The authentication authority sends the *+xns/Session/LoginNotify* message to this URI.

Message: GetSessionURI

+xns/Session/GetSessionURI
Derived from +xns/Core/Message

Obtains the URI for session services provided by this identity. Session messages for this authentication authority (except this one) are posted to this URI. Once an identity name and/or ID has been resolved in XNS, this message is sent to the identity to determine the URI to use for session services.

- **To:** Any identity providing session services (usually authentication authority).
- **From:** Any identity or application in need of session services.

Output

SessionURI *+xns/Core/URI*	URI of the session service for this identity.

Message: Login

+xns/Session/Login
Derived from +xns/Session/Authenticate

Establishes a browser login session, then produces authentication credentials for the session. If a current login session exists, the user will be asked to re-enter her authentication credentials. This message is used if the merchant needs fresh authentication credentials—usually for a sensitive transaction. This message usually originates from the merchant, is sent to a central session service, then forwarded to the authentication authority by the session service. Upon successful login, the authentication authority forwards the user to the central session service with a *+xns/Session/LoginNotify* message.

All arguments required for this message are inherited from the Authenticate message.

- **To:** If specified, this is the identity to send the request to. If unspecified, authentication will be forwarded to the merchant's preferred authentication authority.
- **From:** The originator of the message. This may be the originating merchant, the central session service, or a forwarding AA.

Message: LoginNotify

+xns/Session/LoginNotify
Derived from +xns/Core/Message

Notifies the session service of a successful login. The authentication authority sends this message to the central session service upon successful login. The session service remembers this authentication authority for the browser session, then forwards the user to the merchant URI with a *+xns/Session/SubmitAuthCert* message.

- **To:** The central session service as specified by the SessionURI in the +xns/Session/Login message.
- **From:** The authentication authority.

Input

AuthCert String	Encrypted authentication certificate, Base64 encoded, to be forwarded to the merchant.

Message: LogoutNotify

+xns/Session/LogoutNotify
Derived from +xns/Core/Message

Notifies a merchant that the user has logged out. The user is directed to the LogoutURI specified in the originating *+xns/Session/Authenticate* or *+xns/Session/Login* message. A RedirectURI parameter is appended to the HTTP GET request by the session service. Upon receipt of this request, the merchant should remove any session state associated with this user. The merchant then must redirect the HTTP request to the specified RedirectURI. This is not an XNS message–merely an HTTP GET request to a specified URI at the merchant site. It is described as an XNS message to provide a unique name for this operation and provide a place to contain its specification.

- **To:** This is an HTTP GET request to a merchant.
- **From:** The central session service.

Input

RedirectURI *+xns/Core/URI*	URI to direct the browser after session state is removed.

Message: SubmitAuthCert

+xns/Session/SubmitAuthCert
Derived from +xns/Core/Message

Submits an authentication certificate to a merchant. This message is sent to the merchant who originated a *+xns/Session/ Authenticate* or *+xns/Session/Login* message. The authentication certificate contained in this message was created by the authentication authority for this merchant.

- **To:** The merchant identity.
- **From:** The central session service or authentication authority.

Input

AuthCert String	Encrypted authentication certificate, Base64 encoded, created by the *+xns/Authentication/CertifySession* message for the purpose of certifying a login session. The authentication certificate has been signed by the authentication authority, compressed, then encrypted using the public key of the merchant identity.
AuthIdentity *+xns/Core/IdentityAddress*	Address of the authentication authority responsible for authenticating the user.

Message: XNSSessionLogout

+xns/Session/XNSSessionLogout
Derived from +xns/Core/Message

Invalidates the current authentication session. This message originates from a merchant, is sent to the central session service, then forwarded to the authentication authority. Upon receipt, the central session service sends this message to the authentication authority, and also sends a *+xns/Session/LogoutNotify* message to all merchants except the merchant originating the message.

- **To:** Originally to the central session service; forwarded to the authentication authority.
- **From:** The originating merchant identity's XNS address. *+xns/Session/LogoutNotify* messages will be sent to all merchants participating in the SSO session except this merchant.

Input

RedirectURI *+xns/Core/URI*	URI to redirect the user to upon logout. This is usually a page at the merchant web site which does not require authentication, such as a home page.

Data: ServiceProfile

+xns/Session/ServiceProfile
Derived from +xns/Core/XNSObject

Identity-specific preferences for the session service.

Data Elements

SessionURI *+xns/Core/URI*	The URI of the session service for this identity.

3

XNS Addressing Specification

This chapter describes the standard syntax for addressing data in an XNS identity document, including the five rule sets that normatively define the XNS address format.

Introduction to XNS Addressing

The W3C XPath 1.0 Recommendation [9] establishes a standard syntax for addressing the nodes of a structured XML document. Since XNS is a network of linked XML documents, it has a similar need to standardize the syntax for addressing across a network of XNS identity documents.

However unlike XPath, which was designed primarily for programmatic use (and includes many additional functions for querying data sets within an XML document), XNS addressing needs to fulfill requirements for machine efficiency, human usability, and identity persistence. For a complete discussion of those requirements, please see the XNSORG white paper *From Name Service to Identity Service: How XNS builds on the DNS Model*.

This specification provides the normative requirements for XNS address validity. It includes five rule sets:

1. The EBNF definition of XNS addressing syntax.
2. The EBNF definition for URI encoding of XNS addresses and XNS service invocations as URNs (XRIs).
3. XNS ID normalization rules.
4. XNS Name normalization rules.
5. Higher-order rules for XNS address validity, including validity of the global attribute values used to construct and resolve XNS addresses.

EBNF Definition of XNS Addressing Syntax

Following is the authoritative EBNF definition of an XNS address (see the Notation section of the W3C XML 1.0 Recommendation [6] for a summary of the EBNF syntax). All XNS addressing values used in XNS implementations MUST conform to this EBNF definition.

```
[ 1] XNSAddress ::= XNSID | XNSName | AbsoluteAddress
[ 2] XNSID ::= IdentityID | IdentityDataID | DataID | RelativeDataID
[ 3] IdentityID = ':' HostID ':' [IdentityIDNode *('.' IdentityIDNode)]
[ 4] HostID ::= ('(' URN ')' ['.' OID]) | OID
[ 5] URN ::= Uniform Resource Name as specified in IETF RFC 2141 [1]
[ 6] OID ::= INT *('.' INT)
[ 7] INT ::= Non-negative integer
[ 8] IdentityIDNode ::= ID | '(' IdentityID ')'
[ 9] ID ::= XML character string normalized according to the XNS ID
Normalization Rules
[10] IdentityDataID ::= IdentityID DataID
[11] DataID ::= ';' DataIDNode [RelativeDataID]
[11] RelativeDataID ::= *('.' DataIDNode) [',' Version]
[12] DataIDNode ::= ID | ( '(' IdentityDataID ')' )
[13] Version ::= ('v' VersionNumber) | ('t' VersionDate)
[14] VersionNumber ::= Non-negative integer
[15] VersionDate ::= XML DateTime instance
[16] XNSName ::= IdentityName | IdentityDataName | DataName |
RelativeDataName
[17] IdentityName ::= (NamespaceSymbol | '//') IdentityNameNode
*('/' IdentityNameNode)
[18] NamespaceSymbol ::= ('=' | '@' | '+' )
[19] IdentityNameNode ::= Name | '(' IdentityAddress ')'
[20] Name ::= XML character string normalized according to the XNS Name
Normalization Rules
[21] IdentityAddress ::= (IdentityID ['!' IdentityName]) | IdentityName
[22] IdentityDataName ::= IdentityName DataName
[23] DataName ::= '/' RelativeDataName
[24] RelativeDataName ::= DataNameNode *('/' DataNameNode) ['/,'
Version]
[25] DataNameNode  ::= Name | ( '(' IdentityDataAddress ')' )
[26] IdentityDataAddress ::= (IdentityDataID ['!' IdentityDataName]) |
IdentityDataName
[27] AbsoluteAddress ::= IdentityAddress | IdentityDataAddress
```

Note that the production names used in this EBNF correspond directly to the XNS datatypes defined in XNS Core service. Though Core.XSD defines XML Schema pattern syntax for each datatype, this EBNF is the normative syntax for these datatypes.

Key Concepts in the EBNF

The EBNF is based on a handful of concepts that are repeated throughout the productions. The following sections explain these key concepts prior to the line-by-line documentation.

XNS Objects and Type Collections

Although not represented directly in the EBNF, the most fundamental concept in XNS addressing is the concept of an *XNS object*. If every identity in XNS is modeled as a logical XML document (the identity document), and every attribute of this identity is modeled as a tree of XML nodes within the identity document, then XNSObject is the abstract data type from which every XNS-addressable node in this tree is derived. The XNSObject data schema, defined in XNS Core service, defines the global attributes and elements used for all addressing metadata in XNS. For more information, see Core service.

A second underlying concept not represented directly in the EBNF is *type collections*. This is the special type of XNS object used to store instances of other XNS objects of the same datatype. For more information on the TypeCollection datatype, see Data Service.

Object IDs, Names, and Addresses

Three of the most fundamental requirements of XNS are that it be able to:

1. Provide an abstraction layer capable of representing and modeling the identity of any actor in web services—machine, network location, application, user, business, taxonomy category, etc.,
2. Enable this identity to persist for the lifetime of the identity, and
3. Enable this identity abstraction layer to be federated across any number of XNS identity servers or clients for fully decentralized, delegated identity management.

To meet these requirements XNS follows the architectural principle of *semantic abstraction*—separating non-persistent semantic identifiers (names) from persistent abstract identifiers (IDs). In most computer naming systems, a name is resolved directly to the physical location of a resource—a file on a disk, a host machine on a network, a record in a database. In XNS a name resolves to an XNS ID, which in turn resolves to the network location of the identity document or a node within it. This network location is expressed as a Uniform Resource Identifier (URI) [10]. Since URIs do not require persistence of the address, an XNS address meets the higher persistence requirements of a Uniform Resource Name (URN) [1].

Since the address of an XNS object may use a name, an ID, or both, XNS addressing supports all three concepts:

- **IDs** are persistent non-semantic addressing values intended primarily for machine use. XNS IDs are permanent identifiers that can be either local or global in scope, but which *never change* once they are assigned to an XNS identity or an identity attribute. If the identity or identity attribute is deleted from the system, the ID or IDs used to identify it are retired and never reused. XNS IDs are the basis for all persistent relationships in XNS, whether references or links. XNS IDs are stored in the XNS global ID attribute of an XNS object.

- **Names** are non-persistent addressing values with semantic meaning intended primarily for human use. XNS names represent semantic relationships that can change as real-world identity names and relationships change, so they do not have the same persistence requirements as XNS IDs. XNS naming is implemented as an abstraction layer on top of XNS IDs, i.e., an XNS name resolves to an XNS ID before it is resolve to the network location of the target XNS object (URI). XNS names are stored in the XNS global Name attribute of an XNS object.

- **Addresses** are a composite addressing type that can consist of either an XNS ID or an XNS name or a combination of both. In the latter case the XNS ID is authoritative and the XNS name always serves as a human-readable comment.

Object Versions

In XNS, maintaining state is necessary to support the requirements of being able to uniquely address, share, and synchronize identity attribute values. Identity documents must be able to unambiguously identify different versions of XNS objects representing identity attributes at different moments in time.

In XNS, version metadata for every XNS object is standardized using in two global attributes—IV (Instance Version) and SV (Schema Version)—and two global elements—XNSCreateDate and XNSUpdateDate. To unambiguously address a specific version of an XNS object, the EBNF Version productions allow the version value identifying the target version of an XNS object to be appended to the XNS ID or XNS name.

This solves a longstanding problem with URI syntax: how to maintain the persistent identity of a resource while still being able to authoritatively specify a particular version of that resource. Under current W3C specifications such as P3P, a different URI must be used to specify a different version of a resource such as a privacy policy. This is necessary because URI syntax does not specify a versioning component, so the portion of a URI that must change to reflect version changes can only be established by local convention.

XNS addressing syntax solves this problem by providing an explicit global versioning component. This version value can be in one of two formats:

- **Version Number.** This is an integer representing the version of the XNS object as stored in the IV attribute. Note that there is no requirement that version numbers be sequential; simply that they increase in value. This allows numeric equivalents of the version date to be used as version numbers.
- **Version Date.** This is an XML dateTime instance (as specified by W3C XML Schemas Part 2 [5]) as stored in the XNSUpdateDate element. See the *Version Date Format* rule for more about this format.

Identities and Identity Data

Absolute and relative are concepts that apply to almost any addressing system. Absolute addresses are globally unique and can always be resolved regardless of the current addressing context (i.e., they have a known starting point). By contrast, relative addresses are not globally unique and can only be resolved relative to the current addressing context. In XNS the concepts of absolute and relative are modeled by the concepts of identity and data. An XNS identity is always an absolute identity, capable of being globally independent of any other identity, while any data contained by that identity—the set of XNS objects describing the attributes of the identity or its relationship to other identities—are relative to the identity, since they do not logically make sense outside of that context.

From an addressing perspective this means an XNS identity is conceptually similar to a disk drive in a file system or a network drive in UNIX, while the data contained by an identity is conceptually similar to the files contained on this drive. XNS simply abstracts the concept of "drive" to *any* identifiable container of data—the identity document is the abstract XML representation that top-level container. All nodes below the root node of the identity document represent the attributes (data) contained in this container.

Since all XNS identities are absolute, they require absolute addresses, and registering and resolving these addresses requires inter-identity communications because the addresses span multiple identity documents. By contrast, the address of any data within an identity document is always relative to the root node of that identity document and can therefore be resolved entirely by the authoritative XNS identity, the same way locating a filename on a local disk drive does not require any calls to the outside network.

To represent these concepts, the following four terms are used consistently throughout the EBNF:

- **Identity** is used as the prefix for all absolute values—globally unique IDs, names, or addresses—that resolve to the root node of an identity document.
- **IdentityData** is used as the prefix for all absolute values—globally unique IDs, names, or addresses— that resolve to any lower-level node within an identity document.
- **Data** is used as the prefix for IDs or names that are not globally unique, but unique only relative to the root node of an identity document.
- **RelativeData** is used as the prefix for IDs or names that are unique only relative to the current node of an identity document.

Host Identities and Hosted Identities

An XNS identity can represent any identifiable entity, from a person to a taxonomy category. However because an XNS identity must be represented by an identity document that physically resides somewhere on the network, an XNS identity address must resolve to the network address of this resource.

In XNS this is the job of a special type of identity called a *host identity*. Host identities represent the identity of a network endpoint—a device with a physical network address on which run the XNS identity agents responsible for identity documents. A host identity is simply an XNS identity with at least one known set of attributes: a list of URIs representing different XNS *connectors* (transport protocols) over which this host machine accepts XNS messages.

A host identity can be standalone (self-hosted), or it can host any number of other XNS identities called *hosted identities*. The collection of the host identity and all hosted identities is called a *host community*. Every identity in a host community includes the host identity's address in its own XNS address just like every web page in a web site includes the same base DNS address (e.g., "www.example.com/").

It is important to clarify that a host identity may not be associated with any identity it hosts any more than the operator of a web server is associated with the identity of any web site that runs on it. While a host identity has its own XNS address, and can indeed store and manage any of the attributes that profile the host device or operating environment (including its trust credentials), it may not have anything else in common with the identities it hosts besides being co-located at the same network endpoint.

Cross-references

The final key concept in XNS addressing architecture is one that is critical to XNS as a distributed, federated identity system. Because XNS identities are abstract representations of real-world identities, more than one XNS identity can be used to represent a real-world identity, called a *principal*. For example, the identity of a real-world person may be represented by multiple XNS identities operating in different host communities in different XNS *domains* (spheres of legal ownership). So, for example, a person may have one XNS identity on an identity server operated by her bank, another on an identity server operated by her employer, another on an identity server operated by her doctor, another on an XNS client operating on her laptop, etc.

Each of these identities maintains a physically separate identity document, however they all logically represent the same real-world principal. So not only are the identity documents logically equivalent, but so are many of the XNS objects contained within them (e.g., home phone number, email address, hair color, shoe size, etc.). *Note*: at a particular point in time they may not be *physically equivalent*, as they may not be fully synchronized.

XNS needs the ability to express this logical equivalence so the same identity or identity attribute can be recognized across multiple host communities and domains when the identity controller wants this behavior. (Note that this behavior may be expressly prohibited when an identity controller wishes to remain anonymous or pseudonymous). To support this, XNS addressing includes the concept of *cross-references*. Any XNS object in an identity document (including the document root object representing the identity itself) can be cross-referenced with another logically equivalent XNS object in a different identity document. This is done by including the identity address of the equivalent XNS object in the source XNS object in a list element called XNSXRefs.

Essentially cross-references allow an identity or its attributes to be indexed not just by a local ID and name but also by any number of cross-references to logically equivalent identities or attributes anywhere else on the XNS network. Cross-references are supported syntactically by enclosing them in parentheses. The EBNF productions include the following special terms for the syntax elements where cross-references can be used:

- **IdentityIDNode** is the term used for any XNS object in an XNS ID path that terminates in the root node of an identity document. IdentityIDNodes can be addressed by either a local ID or a cross-reference ID.
- **DataIDNode** is the term used for any XNS object in an XNS ID path that terminates in a node below the root node of an identity document. DataIDNodes can be addressed by either a local ID or a cross-reference ID.
- **IdentityNameNode** is the term used for an XNS object in an XNS name path that terminates in the root node of an identity document. IdentityNameNodes can be addressed by either a local name or a cross-referenced XNS address (either an XNS ID or XNS name).

- **DataNameNode** is the term used for an XNS object in an XNS ID name that terminates in any node below the root node of an identity document. IdentityNameNodes can be addressed by either a local name or a cross-referenced XNS address (either an XNS ID or XNS name).

Line-By-Line Documentation of the EBNF

Using the key concepts explained above, the following sections step through the EBNF productions to explain the structure of an XNS address in detail.

XNS Addresses

```
[ 1] XNSAddress ::= XNSID | XNSName | AbsoluteAddress
```

An XNS address can be one of three overall types. The first two, XNSID and XNSName, are atomic. The third, AbsoluteAddress, is a composite of an XNS ID value or an XNS name value (or both) that forms the absolute address of an XNS identity document or a data node within it.

XNS IDs

```
[ 2] XNSID ::= IdentityID | IdentityDataID | DataID | RelativeDataID
```

As explained in the Object IDs, Names, and Addresses section above, an XNS ID is a permanent semantic identifier of any XNS object. It can be one of four types depending on whether it is the absolute ID for the XNS object at the root node of an identity document (IdentityID), the absolute ID for an XNS object below the root node of an identity document (IdentityDataID), the relative ID of an XNS object in relationship to the root node of an identity document (DataID), or the relative ID for an XNS object in relationship to the current node of an identity document (RelativeDataID). Each of these four types is explained in the following sections.

Identity IDs

```
[ 3] IdentityID = ':' HostID ':' [IdentityIDNode *('.' IdentityIDNode)]
[ 4] HostID ::= ('(' URN ')' ['.' OID]) | OID
[ 5] URN ::= Uniform Resource Name as specified in IETF RFC 2141
[ 6] OID ::= INT *('.' INT)
[ 7] INT ::= Non-negative integer
[ 8] IdentityIDNode ::= ID | '(' IdentityID ')'
[ 9] ID ::= XML character string normalized according to the XNS ID
Normalization Rules
```

An IdentityID is a path that begins with a colon representing the abstract XNS ID community identity. This is followed by a globally unique ID value representing the host identity. Line 4 defines that the host identity portion of an IdentityID can one of two types:

- **A URN** (Uniform Resource Names) is a URI that conforms to the syntax defined in IETF RFC 2141 [1]. URNs provide persistent identifiers for resources whose network locator (URL) may change over time. One common form of a URN uses a UUID, a 36-hex-character string generated according to a known specification so that for all practical purposes it is guaranteed to be globally unique (the probability of collision is infinitesimally small).
- **An OID** (Object IDs) is a dot-delimited path of non-negative integer values that are commonly used in directory systems such as X.500 and LDAP. Each integer in an OID path must be unique relative to its parent node. The first-level integers must be unique relative to the OID root node. In the case of XNS, the OID root is the XNS ID community identity managed by XNSORG.

Technically, XNS ID service for host identities is a URN service as defined by the IETF. As a URN service, it specifies an OID syntax that has the advantage of providing a host identity path that can be traversed for ID resolution purposes. However in line 4, XNS also supports any other URN service conforming to the IETF RFC 2141. This includes UUIDs used as URNs, which can be particularly useful for initial development and testing of XNS identities and host communities because they require no central registration authority.

To escape them from XNS ID normalization, a URN other than an XNS OID is contained within parentheses. Following are two examples of URNs used as IdentityIDs. The first is a UUID-based URN. The second is a URN system called Handle operated by the Corporation for National Research Initiatives (CNRI).

```
:(urn:uuid:5a389ad2-22dd-11d1-aa77-002035b29092)
:(urn:hdl:4263537/4090)
```

Because they are native to XNS, OID-based IdentityIDs tend to be much shorter. Examples:

```
:4    (first-level host identity)
:4.781(second-level host registered with host identity :4)
:4.781.23(third-level host registered with host identity :4.781)
```

Line 4 allows URNs and OIDs to also be combined. A URN can also be used to identify a top-level host identity and OIDs for each of the lower-level host identity. Examples:

```
:(urn:uuid:5a389ad2-22dd-11d1-aa77-002035b29092).781.23
:(urn:hdl:4263537/4090).781.23
```

To address any identity in any host community besides the host identity, the host IdentityID is followed by a second colon followed by the hosted IdentityID. The hosted IdentityID is a path of one or more IdentityID nodes delimited by dots. Note that lines 9 and 10 specify that the IdentityID node of a hosted identity can be any ID value that meets the XNS ID Normalization Rules. These rules are much looser than those for host identities, so while the ID of a hosted identity will

typically be an integer, it may also be any other indexing value including the keys commonly used in SQL databases, LDAP directories, and other data stores. This avoids the need for identity documents to maintain the overhead of mapping XNS IDs to native data store keys.

Examples of IdentityIDs for hosted identities:

```
:4.781:560.73
:4.781.23:hbrown44
:(urn:uuid:5a389ad2-22dd-11d1-aa77-002035b29092).781.23:560.73
:(urn:hdl:4263537/4090):hbrown44
```

Line 8 is also where the EBNF allows an IdentityID node to use a cross-reference as explained in Cross-references, above. For example, this allows an identity in one host community to be addressed by the IdentityID it is known by in another host community (provided the identity controller has given consent for this linkage). To separate it as an opaque indexing value, the cross-reference ID is enclosed in parentheses. Examples:

```
:4.781:(:25.754:38056)
:4.781.23:(25.754:hbrown44)
:(urn:uuid:5a389ad2-22dd-11d1-aa77-002035b29092):(:25.754:38056)
:(urn:hdl:4263537/4090):((urn:hdl:559732/1246):hbrown44)
```

Identity Data IDs

```
[10] IdentityDataID ::= IdentityID DataID
```

An IdentityDataID is simply an IdentityID concatenated with a DataID, explained in the productions below. Examples of IdentityDataIDs:

```
:4.781:560.73;14.3
:4.781.23:hbrown44;14.3
:(urn:hdl:4263537/4090):hbrown44;email.home
```

Data IDs

```
[11] DataID ::= ';' DataIDNode [RelativeDataID]
```

A DataID begins with a semicolon followed by at least one DataIDNode. This can be followed by the optional RelativeDataID of any lower-level XNS object. Standing alone, a DataID is always relative to the root node of the current identity document. To make it absolute, it is combined with an IdentityID to form an IdentityDataID (above). Examples of DataIDs:

```
;14
;14.3
;14.3.7
;14.homePhone
;email.homePhone
```

Note that the last two examples use semantic characters as allowed by the XNS ID Normalization Rules. Although technically legal, this practice is strongly discouraged in XNS implementations because semantic relationships change, but XNS IDs MUST remain immutable for the lifetime of a resource.

Relative Data IDs

```
[11] RelativeDataID ::= *('.' DataIDNode) [',' Version]
[12] DataIDNode ::= ID | ( '(' IdentityDataID ')' )
```

A RelativeDataID is any XNS ID that is relative to a node below the identity document root node. To syntactically distinguish them from XNS names, a RelativeDataID always begins with a dot. It can include any number of DataIDNodes, each delimited with a dot. Examples:

```
.7
.7.29
.7.29.4
.homePhone
```

Line 12 permits any form of a DataID to include a cross-reference at any data ID node. Examples of an IdentityDataID, a DataID, and a RelativeDataID that use cross-references:

```
:4.781:560.73;(:732.41:28558;17).3
;14.(:732.41:28558;17.8)
.7.29.(:732.41:28558;17.8)
```

Versions

```
[13] Version ::= ('v' VersionNumber) | ('t' VersionDate)
[14] VersionNumber ::= Non-negative integer
[15] VersionDate ::= XML DateTime instance
```

As explained in Object Versions, above, any XNS ID path to a data node (an IdentityDataID, DataID, or RelativeDataID) can include a version value to identify a specific version of the XNS object. The version value is appended to the data ID path following a comma, and is prefixed with a “v” for integer format or “t” for XML dateTime format (see Object Versions, above).

Examples of XNS IDs that include version values in both formats:

```
:4.781:560.73;14.3,v3
:4.781:560.73;14.3,v4
;7.29,t2001-03-04T20:15:40Z
;7.29,t2001-06-21T07:33:48Z
```

Note that an identity itself cannot be versioned because pure identity is stateless—it either exists or it doesn't. So only the attributes of an identity can be versioned.

XNS Names

```
[16] XNSName ::= IdentityName | IdentityDataName | DataName |
RelativeDataName
```

As explained in the Object IDs, Names, and Addresses section above, an XNS name is a non-persistent semantic identifier for an XNS object. It can be one of four types depending on whether it is the absolute name for the XNS object at the root node of an identity document (IdentityName), the absolute name for an XNS object below the root node of an identity document (IdentityDataName), the relative name of an XNS object in relationship to the root node of an identity document (DataName), or the relative name for an XNS object in relationship to the current node of an identity document (RelativeDataName).

Identity Names

```
[17] IdentityName ::= (NamespaceSymbol | '//') IdentityNameNode
*('/' IdentityNameNode)
[18] NamespaceSymbol ::= ('=' | '@' | '+' )
[19] IdentityNameNode ::= Name | '(' IdentityAddress ')'
[20] Name ::= XML character string normalized according to the XNS Name
Normalization Rules
```

Because XNS identity names can be used as a human-friendly identity address—a consolidation of all other addressing attributes associated with an identity (phone number, email address, postal address, web address, instant messaging address, etc.)—the design goal is to make XNS name syntax as close to natural human language as possible. The result in line 17 is very similar to the UNIX filename syntax widely used in URIs with four key differences:

1. **Three identity namespace prefix symbols** are supported to indicate the three XNS-defined absolute namespaces (line 18). By comparison with DNS top-level domains (.com, .net, .org, .cc, .tv, etc.), these three identity namespace prefix symbols provide the shortest and simplest possible metadata necessary to establish the global context of an identity name. (See below for more.)

2. **Identity names can contain cross-references** to other identities (line 19). This capability is very useful in federated identity management. (See examples below.)

3. **Namestrings can be any legal XML characters** as defined by the W3C XML 1.0 Recommendation, i.e., they can use the full Unicode character set (see http://www.w3.org/TR/REC-xml#charsets). In addition, the design goal of the normalization rules for identity names is to permit maximum expressiveness while still meeting the minimum requirements for distinguishability of names—see the XNS Name Normalization Rules. This enables XNS identity names to be fully internationalized.

4. **Names for data objects can be versioned** using the same syntax as XNS ID versioning.

An IdentityName is a path that can begin with either: a) one of the three identity namespace prefix symbols ("=", "@", and "+"), or b) a double forward slash ("//") representing the abstract XNS Name community identity managed by XNSORG. The three identity namespace prefix characters are simply shortcuts that expand into the full pathname following the Namespace Symbol Expansion rule, below. These three namespaces represent the three fundamental types of identity controllers in XNS:

- **The Personal namespace (symbol "=", which expands to "//xns/per/")** is reserved for names registered to represent individuals. These names do not have associated intellectual property rights.
- **The Organizational namespace (symbol "@", which expands to "//xns/org/")** is reserved for names registered to represent any form of legal entity that is not an individual—sole proprietorships, partnerships, corporations, non-profits, governments, academic institutions, etc. Organizational names, also called *business names*, have associated intellectual property rights.
- **The General namespace (symbol "+", which expands to "//xns/gen/")** is reserved for generic names that represent concepts or objects defined by the general public. In trademark law, generic names used in a generic context do not have associated intellectual property rights. XNSORG or its delegate acting as a public trustee registers generic names in the XNS general namespace.

Note that in parsing, a namespace symbol is NOT considered a character in an XNS name value. The namespace symbol is expanded to its corresponding name path, and parsing continues with the subsequent XNS name value. Thus a namespace symbol character used as the literal first character of an XNS name must be escaped. See the XNS Name Normalization Rules, below.

Following the identity namespace symbol or path is at least one XNS name string, which is any set of XML characters normalized according to the XNS Name Normalization Rules, below. This can be followed by any number of additional XNS namestrings, each delimited by forward slashes.

Examples of XNS personal identity names using both namespace symbols and their expanded equivalents:

```
=John
//xns/per/John
=John Smith      (normalizes to johnsmith)
//xns/per/John Smith
=John Smith, Jr. (normalizes to johnsmithjr)
//xns/per/John Smith, Jr.
```

Examples of XNS business identity names:

```
@Example
@Example/Computers
@Example/Computers/Internet
@Smith & Jones       (normalizes to smithjones)
@John Smith Inc.     (normalizes to johnsmithinc)
//xns/org/John Smith, Inc.
```

Note that in the second an third examples above, the identity names are hierarchical: the identity @Example has registered the name "Computers" for another identity, and that identity has registered the name "Internet" for a third identity. Identity names can be hierarchical to any depth.

Examples of XNS general identity names:

```
+xns
+plumber
+Dominican Republic    (normalizes to dominicanrepublic)
//xns/gen/Dominican Republic
```

Examples of identity names using international character sets:

```
=José Villegas, Jr.
@A La François
```

Examples of identity names using cross-references:

```
@Example/(=John Smith)
@Smith & Jones/(+garden rakes)
=John Smith/(+email)
```

As noted above, IdentityNames in XNS do not allow a version value to be appended because identity itself does not have state.

Identity Addresses

```
[21] IdentityAddress ::= (IdentityID ['!' IdentityName]) | IdentityName
```

Line 19 in the EBNF allows a cross-reference to be not just another IdentityName, but an IdentityAddress. An IdentityAddress is any combination of an IdentityID and an IdentityName that absolutely identifies an identity. If an IdentityID is present, then the IdentityName is optional and delimited by a bang sign ("!") to indicate that it is only a human-readable comment—the Identity ID is always authoritative. This is useful for many contexts (e.g., web pages, software programs, reference manuals, etc.) where the persistence of an identity ID path is needed yet it is also desirable for it to be human readable without performing resolution.

If an IdentityID is not present, then an IdentityAddress must contain an IdentityName, which will be resolved to the authoritative IdentityID. Examples of IdentityAddresses:

```
:230.59:4.13.7421!=John Smith, Jr.
:(urn:uuid:5a389ad2-22dd-11d1-aa77-002035b29092)!@Smith & Reilly
:3.896324!+plumber
```

Note that since an XNS identity controller may register multiple names for an identity, there may be more than one authoritative identity name to use with an identity address. The choice of identity name must be made by the address author.

Identity Data Names

```
[22] IdentityDataName ::= IdentityName DataName
```

As with IdentityDataIDs, an IdentityDataName is simply an IdentityName concatenated with a DataName, explained in the productions below. Examples of IdentityDataNames:

```
=John Smith, Jr./Email/Home
@Example/Computers/Internet/FTP
+plumber/Hourly Rate
```

Note that in the second example above, it is ambiguous whether any name after "@Example" (i.e., "Computers", "Internet", or "FTP") is an identity name or a data name. Only by resolving the name to the underlying XNS ID can it be determined whether the target node is an identity node or a data node.

Data Names

```
[23] DataName ::= '/' RelativeDataName
```

Standing alone, a DataName is always relative to the root node of the current identity document. Like a UNIX filename that is relative to the root directory of the current drive, a DataName always begins with a single forward slash followed by a RelativeDataName. To make it absolute, a DataName is prefixed by with an IdentityName to form an IdentityDataName. Examples of DataNames:

```
/Family/Father's side/Uncles/John
/Uncles/John
/John
```

Relative Data Names

```
[24] RelativeDataName ::= DataNameNode *('/' DataNameNode) ['/,'
Version]
[25] DataNameNode  ::= Name | ( '(' IdentityDataAddress ')' )
```

RelativeDataNames are just like relative path names in UNIX with the exception of the richer XML character set and the ability to include cross-references and version metadata. RelativeDataNames do not have any leading delimiter and use forward slashes to delimit name nodes. Examples:

```
Father's side/Uncles/John
Uncles/John
John
```

To match the same capability in XNS IDs, an XNS data name can address a specific version of a data object by appending the same versioning syntax after a final forward slash: a comma, followed by "v" for an integer version value or "t" for an XML time instant. Examples:

```
/Family/Father's side/Uncles/John/Phone,v3
John/Phone,v4
@Smith & Jones/Inventory/(+garden rakes)/,t2001-03-04T20:15:40Z
=John Smith Jr./Phone/Work/,t2001-06-21T07:33:48Z
```

Line 25 specifies that data names can also incorporate cross-references which themselves can be IdentityDataAddresses (below). Examples:

```
@Yahoo/Computers/Internet/(@IBM/Computers/AS400)
@Smith & Reilly/Tools/(+garden rakes/price)
=John Smith/Friends/(=Mary Frank/Email/Home)
```

Identity Data Addresses

```
[26] IdentityDataAddress ::= (IdentityDataID ['!' IdentityDataName]) |
IdentityDataName
```

Like cross references in identity name nodes (line 19), a cross-reference in a data name node needs to be able to include either an IdentityDataID or an IdentityDataName. Similar to an IdentityAddress (line 21), an IdentityDataAddress can any combination of an IdentityDataID and an IdentityDataName that absolutely identifies a data node within an identity. If an IdentityDataID is present, then the IdentityDataName is optional and the bang sign ("!") indicates that it is only a human-readable comment. If an IdentityDataID is not present, then an IdentityDataAddress must contain an IdentityDataName, which will be resolved to the authoritative IdentityDataID. Examples:

```
:230.59:4.13.7421;14.2!=John Smith, Jr./Email/Work
:(urn:hdl:4263537/4090):custACME;AEFF3CB.3956!@Acme/Eastern/Boats/
:3.896324:2499;77.98103!+plumber/(+flood repair)/Zip Code/98103
```

Absolute Addresses

```
[27] AbsoluteAddress ::= IdentityAddress | IdentityDataAddress
```

Lastly, an AbsoluteAddress is a composite datatype allowing either an IdentityAddress or an IdentityDataAddress. This datatype is useful for specifying an XNS address that must be absolute but can be either an XNS ID or XNS name and can resolve to either an identity node or a data node within an identity.

XRI (XNS Resource Identifier) EBNF Definition

One of the primary purposes of XNS addressing is to provide URNs [1]—permanent references to resources that will not change for the lifetime of the resource. In order for an XNS address to function as a URN, it must first be specified in a URI (Uniform Resource Identifier) [10] format. This type of a URI is called an XRI (XNS Resource Identifier).

XRIs also include the capability to invoke an XNS service associated with the target resource just as URIs using the HTTP schema can include query parameters following a question mark. XRIs use the same question mark syntax.

When an XNS address is encoded as a URI according to IETF RFC 2396 [10] in an XNS implementation, it MUST conform to the following EBNF definition.

```
[01] XRI ::= URIScheme (ServiceCall | IdentityAddress |
IdentityDataAddress)
[02] URIScheme ::= HTTP | URN | XNS
[03] HTTP ::= 'http://' XNSResolverHostAddress '/xns:'
[04] XNSResolverHostAddress ::= DNS or IP address of XNS resolver host
[05] URN ::= 'urn:xns:'
[06] XNS ::= 'xns:'
[07] ServiceCall ::= '?' MessageAddress '(' [Argument] *(';' Argument)
')'
[08] MessageAddress ::= IdentityDataAddress of XNS message definition
[09] Argument ::= Argument to the XNS message
[10] IdentityAddress ::= as defined in XNS Address EBNF
[11] IdentityDataAddress ::= as defined in XNS Address EBNF
```

The three URI prefixes correspond to the three URI schemes [10] that will most commonly be used with XNS addresses:

- **The HTTP scheme** is used to redirect XNS address resolution requests to a resolver available at a DNS or IP address, i.e., any address recognized by the HTTP URI scheme.
- **The URN scheme** is used to redirect XNS address resolution requests to a URN service [1] that understands XNS address resolution.

- **The XNS scheme** is the native URI scheme for XNS, and presumes the URI parser understands XNS addressing.

Note that in the HTTP and URN schemes, it is the native XNS URI scheme "xns:" that delimits the start of the XNS address string. In the HTTP scheme, these must be the first four characters following the forward slash that terminates the XNS resolver host address.

XNS ID Normalization Rules

The XNS Address EBNF establishes the strict syntax for the IDs used for host identities (they must be either OIDs or URNs). However the global rules for legal characters and normalization of XNS ID values at the identity or identity attribute (data) levels are intended to be looser to permit the use of a wide variety of conventional database keys, and to also allow identity controllers to establish their own stricter normalization rules for specific ID spaces.

The question of what are the optimal ID normalization rules to impose on all XNS implementations is one on which XNSORG seeks community feedback through the forums available at www.xns.org. As described in Future Work, the result will be a formal EBNF definition for XNS ID normalization. Until then the following top-level rules are normative:

Legal XML Characters in IDs

A normalized XNS ID value MUST NOT include any character defined as an illegal character by the W3C XML 1.0 Recommendation [6].

Unambiguous IDs

A normalized XNS ID value MUST NOT include any character which causes ambiguity in parsing the ID value according to the EBNF definition of XNS addressing syntax.

XNS Name Normalization Rules

Because it involves human semantics, internationalization, and the Unicode character set, XNS name normalization is a much more complex subject than ID normalization. Again the intention is to establish a baseline set of global rules for all implementations that can be further restricted within delegated namespaces. For the namespaces under its governance, the ultimate goal of XNSORG is to define name normalization rules that would identically normalize namestrings that a typical speaker of the relevant language would consider semantically equivalent.

XNSORG invites community feedback on composing the XNS name normalization rules through the forums available at www.xns.org. As described in Future Work, the result will be a formal EBNF definition for XNS Name normalization. Until then, the following top-level rules are normative:

Legal XML Characters in Names

A normalized XNS Name value MUST NOT include any character defined as an illegal character by the W3C XML 1.0 Recommendation [6].

Unambiguous Names

A normalized XNS Name value MUST NOT include any character which causes ambiguity in parsing the name value according to the EBNF definition of XNS addressing syntax.

XML Letters and Digits

A normalized XNS Name value MUST NOT include any character that is not classified as either a Letter or Digit according to Appendix B of the W3C XML 1.0 Recommendation [6].

Escape Character

The ASCII character 092 decimal (backslash "\") MUST be used to escape any character used in an XNS Name value which would not otherwise allowed by the XNS name normalization rules, including this character itself.

XNS Validity Rules

The XNS Base Service Specification establishes in WSDL and XSD formats the definitions of the base set of XNS services and the datatypes on which they operate. These formats permit validation of XNS messages by the XML validity rules specified in the W3C XML Schemas Recommendation [4]. In addition the previous section specifies the EBNF rules governing XNS address syntax. However because XNS is a network of interlinked XML documents, there are a handful of higher-order validity rules that cannot be expressed in either XSD, WSDL, or EBNF formats because they depend on the valid use of XNS addresses for references or links.

An identity document or message meeting these rules is referred to as being *XNS valid*—a superset of being XML valid. A software program (such as an XNS identity server or client implementation) that can confirm XNS validity is referred to as an *XNS processor*. This section provides the rules for XNS processors to determine XNS validity.

XNS validity is based primarily on the use of XNS addresses to resolve references to the XNS definition objects that are the source for the WSDL or XSD definitions needed to perform XML validation. An XNS processor can obtain the normative WSDL definition of an XNS service, called the *source service definition*, by sending an XNS Discovery DescribeService message to the XNS address of the source ServiceDef object. (The ServiceDef object is defined in XNS Discovery Ser-

vice—see *+xns/Discovery.)* The same message can be used to obtain the XSD definition of an XNS datatype, called the *source schema definition*, from the source DataTypeDef object (a subelement of a ServiceDef object).

Note that because XNS includes explicit support for schema evolution, XNS validity can only be defined relative to a schema version at a particular point in time. When an XNS element is first created, this point in time is captured using the XNSCreateDate subelement of XNSObject (defined in Core service). This value is called the *creation date*. The time of the most recent update to the element is captured using the XNSUpdateDate subelement of XNSObject; this value is called the *update date*.

Updates to the value of an XNS element may trigger validation against an updated version of the schema. Upversioning the schema instance may require the identity storing the instance to perform XSL transformations. The versioning components of XNS addressing syntax are defined in the XNS Address EBNF Definition section.

A final note regarding validity: it can only be performed when *the XNS processor is within the addressing community of the identity document being validated.* An addressing community is a set of identities whose inter-registration of IDs and names (whether via hierarchical or network topologies) makes their XNS addresses resolvable by the other identities within the community. The peer-to-peer architecture of XNS makes possible any number of addressing communities, and identity validity can only hold within a community that has access to the source definitions against which the instances must be validated.

The special XNS global addressing attributes of the XNSObject schema, defined in XNS Core service, and the XNS validity rules in which they participate can be summarized as follows:

Attribute Name	Short For	Datatype	Purpose	Participates in XNS Validity Rules
ID	—	String	Stores the EBNF ID value (line 9)	ID Uniqueness ID Persistence
Name	—	String	Stores the EBNF Name value (line 20) in the context of a parent object	Name Uniqueness
IName	Independent Name	String	For a child object, stores the EBNF Name value (line 20) in the context of the TypeCollection object under which it is stored (see Data Service).	—
PI	Promote ID	Boolean	Establishes the context for ID uniqueness	ID Uniqueness Text List IDs and Names

Attribute Name	Short For	Datatype	Purpose	Participates in XNS Validity Rules
PN	Promote Name	Boolean	Establishes the context for name uniqueness	Name Uniqueness Text List IDs and Names
IV	Instance Version	String	Stores the instance version value	Version Number Format
SV	Schema Version	String	Stores the schema version value	Datatype Version Validity Version Date Format
AV	Auto Version	Boolean	Establishes whether all changes to the object should be automatically versioned	—
XA	XNS Address	XNS Address	Stores the absolute ID of the parent node	Fragment Portability

Following are the XNS validity rules. Note that if a rule cannot be validated against an identity document directly by an XNS processor, it is considered operational, i.e., it must be validated via human review or software audit.

ID Uniqueness

The value of an ID attribute MUST be unique in the ID space of the parent element. If the value of the PI (Promote ID) attribute of the parent element is "TRUE", then the value of the ID attribute MUST be unique in the ID space of the parent of the parent element.

This rule ensures the uniqueness of ID values within the desired context.

ID Persistence

An XNS ID value, once assigned by any identity to any XNS resource, MUST NOT change and MUST NOT be reused or reassigned. If the target identity or target node is terminated from the XNS network, the ID MUST be retired using the "xns:+xns/ID/Retire" message in XNS ID service and never reused.

This rule ensures any ID value used in XNS will always, in the context of the assigning element or identity, remain a unique identifier of the resource to which it was originally assigned.

Name Uniqueness

The value of a Name attribute MUST be unique in the XNS namespace of the parent element. If the value of the PN (Promote Name) attribute of the parent element is "TRUE", then the value of the name attribute MUST be unique in the namespace of the parent of the parent element.

This rule ensures the uniqueness of XNS names within the desired context.

Fragment Portability

The root element of any XNS identity document fragment MUST include an authoritative absolute XNS ID of the root node as the value of the XA (XNS address) attribute.

This rule ensures that a global context for any identity document fragment can be maintained when the document fragment is transported outside the context of its containing identity document. Note that because identity documents and data nodes may have more than one XNS ID, there may more than one legal value of the XA attribute. This ensures that XNS can support anonymity and pseudonymity as well as veronymity (true identity).

List Elements

For all XNS datatype definitions, an XML list element for storing a list of instances of this datatype MUST be defined and its element name MUST be the datatype element name concatenated with the string "List".

This rule ensures that all implementations know the element name of the list element associated with any datatype.

Text List IDs and Names

In all elements of the datatype TextList or derived from the datatype TextList (the list element for the datatype Text defined in Core service), the value both the PI and PN attributes MUST be "TRUE".

This rule ensures that all implementations can correctly navigate text lists.

Version Number Format

In the XNS EBNF, a Version value in the "v" (number) format used to represent a version number MUST be equal to the value of the "IV" (Instance Version) attribute of the target XNS object.

This rule ensures that numeric-based version values can be resolved to a specific version of the target object.

Version Date Format

In the XNS EBNF, a Version value in the "t" (time) format used to represent a version date MUST be an instance of the XML dateTime simple datatype (http://www.w3.org/2001/XMLSchema#int) as defined in the W3C XML Schemas 1.0 Part 2 Recommendation [5]. In addition, this value MUST use Coordinated Universal Time (UTC) and thus MUST include a capital "Z" as the terminating character of the dateTime string. This version value MUST be equal to the value of the XNSUpdateDate element of the target XNS object.

This rule ensures that time-based version values use the same format and can be resolved to a specific version of the target object.

XNS Reserved Namespace

The absolute namespace //xns/ is reserved and MUST be used only as specified by the XNS Public Trust Organization, which manages this namespace on behalf of the XNS community.

This rule ensures that there is at least one globally interoperable namespace for addressing and cross-referencing supported across all XNS implementations.

Namespace Symbol Expansion

In the XNS EBNF, the namespace symbol "=" MUST be expanded to the name path "//xns/per/"; the namespace symbol "@" MUST be expanded to the name path "//xns/org/"; the namespace symbol "+" MUST be expanded to the name path "//xns/gen/". This expansion MUST be performed before applying EBNF parsing rules to the XNS name following the namespace symbol.

This rule ensures that namespace symbols used in XNS identity names are interpreted correctly by XNS parsers.

Cross-reference Resolution

If an XNS object contains a cross-reference in its XNSXRefs list, the identity containing that XNS object MUST be able to resolve that cross-reference to the contained XNS object starting from to the root node of the identity document.

This rule ensures that cross-references will be resolved consistently by all XNS implementations.

Type Collection Cross-references

All XNS type collection objects (see XNS Data Service) MUST contain a cross-reference to the logically equivalent type collection object in the identity authoritative for the type definition. The authoritative IdentityDataAddress for this type collection object MUST be published as the value of the TypeCollectionAddress subelement of the DataTypeDef subelement of the ServiceDef object containing this definition.

This rule ensures that there is a known cross-reference for any type collection that can be resolved by all XNS identities containing that type collection.

Datatype Validity

All XNS elements derived from the abstract schema XNSObject defined in XNS Core service MUST have a "type" attribute whose value is the authoritative IdentityDataAddress of the source DataTypeDef object, and the element MUST be a valid instance of this schema definition. The authoritative IdentityDataAddress MUST be published as the value of the DataTypeDefAddress subelement of the DataTypeDef subelement of the ServiceDef object containing this definition.

This rule ensures that an XNS processor can locate the XNS datatype definition necessary to validate an XNS element instance.

Datatype Version Validity

If an XNS element does not have an "SV" (Schema Version) attribute or if the "SV" value is null, the element MUST be valid according to the version of the source schema definition current as of the element's creation date. If an XNS element instance is updated to conform to an updated version of the schema definition, it MUST have an "SV" attribute and the value of this attribute MUST be equal to the value of the SV attribute of the updated schema definition.

This rule ensures that an XNS processor can locate the correct version of the XNS datatype definition necessary to validate an XNS element instance. See the XNS Address EBNF Definition for more details about the "SV" addressing attribute and XNS addressing version attributes.

Message Validity

The XNS Message root element MUST have a "type" attribute whose value is the authoritative IdentityDataAddress of the source MessageDef, and the message MUST be a valid instance of the message defined by this service definition. The authoritative IdentityDataAddress MUST be published as the value of the MessageDefAddress subelement of the MessageDef subelement of the ServiceDef object containing this definition.

This rule ensures an XNS processor can locate the XNS service definition necessary to validate an XNS message.

4

Future Work

This chapter explores future directions for the XNS Technical Specifications. Your feedback is invited.

Introduction

This document represents the first published version of the XNS Technical Specifications. Part of the charter of XNSORG is to oversee future evolution of the XNS Technical Specifications through public feedback and input. This work may occur at XNSORG or may be delegated to other technical standards bodies.

Currently XNSORG has identified the following areas for future work:

Addressing Specification

Two areas of the XNS Addressing Specification need further definition:

- **EBNF Definition of the XNS Name Normalization Rules.** As explained in that section of the specification, normalization for Unicode text strings is a complex problem requiring expertise in internationalization, semantics, trademark law, and other related areas. This work will be ongoing in the XNSORG forums at www.xns.org.
- **EBNF Definition of the XNS ID Normalization Rules.** This problem is less complex than semantic name normalization but needs to consider trade-offs in reserved characters versus ease of integrating with widely deployed database and directory indexing syntaxes. This work will be also be pursued in the forums at www.xns.org.

XNS Bindings

XNSORG is responsible for specifying bindings of the XNS service definitions to industry standard formats, including:

- **XNS to WSDL Service Definition Binding.** A specification is needed for the rules governing implementation of the XNS DescribeService message to generate a WSDL [3] definition file from an XNS service definition object. This should be timed to coincide with release of the WSDL 1.2 specifications from the W3C.
- **XNS to XSD Schema Definition Binding.** The same work applies to generation of an XSD [4] definition file from an XNS service definition object. This work should be timed to employ the XSD 1.1 specification from the W3C.

Additional Base Services

This specification defines three additional services, Directory, Reputation, and Introduction, that are to be included in the XNS base services. This work will be ongoing at www.xns.org. Other candidates for base services include:

- **Notification Service** (also called Alerts) would standardize the vocabulary and messages used to alert an identity controller about identity events and transactions.
- **Event Service** would standardize the interface for scheduling and polling identity events.

Application Services

The XNS base services are a platform for defining higher-level identity services similar to how a operating system is a platform for developing applications. These *application services* can be defined by anyone using XNS Discovery service. XNSORG's charter is to develop (or delegate to other standards bodies to develop) specifications for broad horizontal application services that serve the majority of the XNS community. Currently under consideration are:

- **Contact Service** for sharing and linking of personal and business contact data (e-business cards).
- **Wallet Service** for exchange of digital payment credentials.
- **Email Permission Filtering Service** that employs XNS identity addresses and credentials to provide a new solution to the age-old problem of spam.

Readers of this specification are encouraged to use the forums available at the XNSORG web site (www.xns.org) to provide feedback about the application services they would most like to see XNSORG devote its resources to developing.

APPENDIX A

WSDL Files (Normative)

This Appendix includes WSDL files for each of the ten XNS base services that have messages. As Core Service does not contain any messages of its own, it has no corresponding WSDL files. These files are normative. Machine-readable versions of these files can be downloaded from www.xns.org.

Service: Authentication

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="Authentication"
  targetNamespace="xns:+xns/wsdl/Authentication"
  xmlns:tns="xns:+xns/wsdl/Authentication"
  xmlns:msgs="xns:+xns/Authentication"
  xmlns:core="xns:+xns/Core"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    WSDL abstract service definition for the Authentication XNS service.
  </wsdl:documentation>
  <wsdl:import namespace="xns:+xns/Core"
    location="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
  <wsdl:import namespace="xns:+xns/wsdl/Authentication"
    location="http://specs.onename.com/specs/xns/gen/xns/Authentication.xsd"/>

<wsdl:message name="CertifySessionRequest">
  <wsdl:part name="body" type="msgs:CertifySessionRequest"/>
</wsdl:message>
<wsdl:message name="CertifySessionResponse">
  <wsdl:part name="body" type="msgs:CertifySessionResponse"/>
</wsdl:message>
<wsdl:message name="ChangeCredentialsRequest">
  <wsdl:part name="body" type="msgs:ChangeCredentialsRequest"/>
</wsdl:message>
<wsdl:message name="ChangeCredentialsResponse">
  <wsdl:part name="body" type="msgs:ChangeCredentialsResponse"/>
</wsdl:message>
<wsdl:message name="GetWebLoginURIRequest">
  <wsdl:part name="body" type="msgs:GetWebLoginURIRequest"/>
</wsdl:message>
<wsdl:message name="GetWebLoginURIResponse">
  <wsdl:part name="body" type="msgs:GetWebLoginURIResponse"/>
</wsdl:message>
<wsdl:message name="LoginRequest">
  <wsdl:part name="body" type="msgs:LoginRequest"/>
</wsdl:message>
<wsdl:message name="LoginResponse">
  <wsdl:part name="body" type="msgs:LoginResponse"/>
</wsdl:message>
<wsdl:message name="LogoutRequest">
  <wsdl:part name="body" type="msgs:LogoutRequest"/>
</wsdl:message>
<wsdl:message name="LogoutResponse">
  <wsdl:part name="body" type="msgs:LogoutResponse"/>
</wsdl:message>
<wsdl:portType name="Authentication">
  <wsdl:operation name="CertifySession">
    <wsdl:input message="tns:CertifySessionRequest"/>
    <wsdl:output message="tns:CertifySessionResponse"/>
  </wsdl:operation>
  <wsdl:operation name="ChangeCredentials">
    <wsdl:input message="tns:ChangeCredentialsRequest"/>
```

```
    <wsdl:output message="tns:ChangeCredentialsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetWebLoginURI">
    <wsdl:input message="tns:GetWebLoginURIRequest"/>
    <wsdl:output message="tns:GetWebLoginURIResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Login">
    <wsdl:input message="tns:LoginRequest"/>
    <wsdl:output message="tns:LoginResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Logout">
    <wsdl:input message="tns:LogoutRequest"/>
    <wsdl:output message="tns:LogoutResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Authentication" type="Authentication">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="CertifySession">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:CertifySessionRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Authentication"/>
    </wsdl:input>
    <wsdl:output message="tns:CertifySessionResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Authentication"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="ChangeCredentials">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:ChangeCredentialsRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Authentication"/>
    </wsdl:input>
    <wsdl:output message="tns:ChangeCredentialsResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Authentication"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetWebLoginURI">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GetWebLoginURIRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Authentication"/>
    </wsdl:input>
    <wsdl:output message="tns:GetWebLoginURIResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Authentication"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Login">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:LoginRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Authentication"/>
    </wsdl:input>
    <wsdl:output message="tns:LoginResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Authentication"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Logout">
    <soap:operation soapAction="" style="document"/>
```

```
    <wsdl:input message="tns:LogoutRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Authentication"/>
    </wsdl:input>
    <wsdl:output message="tns:LogoutResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Authentication"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Service: Certification

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="Certification"
  targetNamespace="xns:+xns/wsdl/Certification"
  xmlns:tns="xns:+xns/wsdl/Certification"
  xmlns:msgs="xns:+xns/Certification"
  xmlns:core="xns:+xns/Core"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    WSDL abstract service definition for the Certification XNS service.
  </wsdl:documentation>
  <wsdl:import namespace="xns:+xns/Core"
    location="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
  <wsdl:import namespace="xns:+xns/wsdl/Certification"
    location="http://specs.onename.com/specs/xns/gen/xns/Certification.xsd"/>

<wsdl:message name="AuthorizeRequest">
  <wsdl:part name="body" type="msgs:AuthorizeRequest"/>
</wsdl:message>
<wsdl:message name="AuthorizeResponse">
  <wsdl:part name="body" type="msgs:AuthorizeResponse"/>
</wsdl:message>
<wsdl:message name="ConfirmAttrCertRequest">
  <wsdl:part name="body" type="msgs:ConfirmAttrCertRequest"/>
</wsdl:message>
<wsdl:message name="ConfirmAttrCertResponse">
  <wsdl:part name="body" type="msgs:ConfirmAttrCertResponse"/>
</wsdl:message>
<wsdl:message name="ConfirmRevocationRequest">
  <wsdl:part name="body" type="msgs:ConfirmRevocationRequest"/>
</wsdl:message>
<wsdl:message name="ConfirmRevocationResponse">
  <wsdl:part name="body" type="msgs:ConfirmRevocationResponse"/>
</wsdl:message>
<wsdl:message name="GenerateKeyPairRequest">
  <wsdl:part name="body" type="msgs:GenerateKeyPairRequest"/>
</wsdl:message>
<wsdl:message name="GenerateKeyPairResponse">
  <wsdl:part name="body" type="msgs:GenerateKeyPairResponse"/>
</wsdl:message>
<wsdl:message name="GetCACertsRequest">
  <wsdl:part name="body" type="msgs:GetCACertsRequest"/>
</wsdl:message>
<wsdl:message name="GetCACertsResponse">
  <wsdl:part name="body" type="msgs:GetCACertsResponse"/>
</wsdl:message>
<wsdl:message name="GetCRLRequest">
  <wsdl:part name="body" type="msgs:GetCRLRequest"/>
</wsdl:message>
<wsdl:message name="GetCRLResponse">
  <wsdl:part name="body" type="msgs:GetCRLResponse"/>
</wsdl:message>
<wsdl:message name="GetPublicKeysRequest">
```

```
  <wsdl:part name="body" type="msgs:GetPublicKeysRequest"/>
</wsdl:message>
<wsdl:message name="GetPublicKeysResponse">
  <wsdl:part name="body" type="msgs:GetPublicKeysResponse"/>
</wsdl:message>
<wsdl:message name="RequestAttrCertRequest">
  <wsdl:part name="body" type="msgs:RequestAttrCertRequest"/>
</wsdl:message>
<wsdl:message name="RequestAttrCertResponse">
  <wsdl:part name="body" type="msgs:RequestAttrCertResponse"/>
</wsdl:message>
<wsdl:message name="RevokeCertRequest">
  <wsdl:part name="body" type="msgs:RevokeCertRequest"/>
</wsdl:message>
<wsdl:message name="RevokeCertResponse">
  <wsdl:part name="body" type="msgs:RevokeCertResponse"/>
</wsdl:message>
<wsdl:message name="SubmitForAttrCertConfirmRequest">
  <wsdl:part name="body" type="msgs:SubmitForAttrCertConfirmRequest"/>
</wsdl:message>
<wsdl:message name="SubmitForAttrCertConfirmResponse">
  <wsdl:part name="body" type="msgs:SubmitForAttrCertConfirmResponse"/>
</wsdl:message>
<wsdl:message name="VerifyCertRequest">
  <wsdl:part name="body" type="msgs:VerifyCertRequest"/>
</wsdl:message>
<wsdl:message name="VerifyCertResponse">
  <wsdl:part name="body" type="msgs:VerifyCertResponse"/>
</wsdl:message>
<wsdl:message name="VerifySignatureRequest">
  <wsdl:part name="body" type="msgs:VerifySignatureRequest"/>
</wsdl:message>
<wsdl:message name="VerifySignatureResponse">
  <wsdl:part name="body" type="msgs:VerifySignatureResponse"/>
</wsdl:message>
<wsdl:portType name="Certification">
  <wsdl:operation name="Authorize">
    <wsdl:input message="tns:AuthorizeRequest"/>
    <wsdl:output message="tns:AuthorizeResponse"/>
  </wsdl:operation>
  <wsdl:operation name="ConfirmAttrCert">
    <wsdl:input message="tns:ConfirmAttrCertRequest"/>
    <wsdl:output message="tns:ConfirmAttrCertResponse"/>
  </wsdl:operation>
  <wsdl:operation name="ConfirmRevocation">
    <wsdl:input message="tns:ConfirmRevocationRequest"/>
    <wsdl:output message="tns:ConfirmRevocationResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GenerateKeyPair">
    <wsdl:input message="tns:GenerateKeyPairRequest"/>
    <wsdl:output message="tns:GenerateKeyPairResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetCACerts">
    <wsdl:input message="tns:GetCACertsRequest"/>
    <wsdl:output message="tns:GetCACertsResponse"/>
  </wsdl:operation>
```

```
  <wsdl:operation name="GetCRL">
    <wsdl:input message="tns:GetCRLRequest"/>
    <wsdl:output message="tns:GetCRLResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetPublicKeys">
    <wsdl:input message="tns:GetPublicKeysRequest"/>
    <wsdl:output message="tns:GetPublicKeysResponse"/>
  </wsdl:operation>
  <wsdl:operation name="RequestAttrCert">
    <wsdl:input message="tns:RequestAttrCertRequest"/>
    <wsdl:output message="tns:RequestAttrCertResponse"/>
  </wsdl:operation>
  <wsdl:operation name="RevokeCert">
    <wsdl:input message="tns:RevokeCertRequest"/>
    <wsdl:output message="tns:RevokeCertResponse"/>
  </wsdl:operation>
  <wsdl:operation name="SubmitForAttrCertConfirm">
    <wsdl:input message="tns:SubmitForAttrCertConfirmRequest"/>
    <wsdl:output message="tns:SubmitForAttrCertConfirmResponse"/>
  </wsdl:operation>
  <wsdl:operation name="VerifyCert">
    <wsdl:input message="tns:VerifyCertRequest"/>
    <wsdl:output message="tns:VerifyCertResponse"/>
  </wsdl:operation>
  <wsdl:operation name="VerifySignature">
    <wsdl:input message="tns:VerifySignatureRequest"/>
    <wsdl:output message="tns:VerifySignatureResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Certification" type="Certification">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="Authorize">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:AuthorizeRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:AuthorizeResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="ConfirmAttrCert">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:ConfirmAttrCertRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:ConfirmAttrCertResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="ConfirmRevocation">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:ConfirmRevocationRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:ConfirmRevocationResponse">
```

```
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GenerateKeyPair">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GenerateKeyPairRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:GenerateKeyPairResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetCACerts">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GetCACertsRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:GetCACertsResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetCRL">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GetCRLRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:GetCRLResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetPublicKeys">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GetPublicKeysRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:GetPublicKeysResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="RequestAttrCert">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RequestAttrCertRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:RequestAttrCertResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="RevokeCert">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RevokeCertRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:RevokeCertResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
```

```
  </wsdl:operation>
  <wsdl:operation name="SubmitForAttrCertConfirm">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:SubmitForAttrCertConfirmRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:SubmitForAttrCertConfirmResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="VerifyCert">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:VerifyCertRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:VerifyCertResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="VerifySignature">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:VerifySignatureRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:input>
    <wsdl:output message="tns:VerifySignatureResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Certification"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Service: Data

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="Data"
  targetNamespace="xns:+xns/wsdl/Data"
  xmlns:tns="xns:+xns/wsdl/Data"
  xmlns:msgs="xns:+xns/Data"
  xmlns:core="xns:+xns/Core"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    WSDL abstract service definition for the Data XNS service.
  </wsdl:documentation>
  <wsdl:import namespace="xns:+xns/Core"
    location="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
  <wsdl:import namespace="xns:+xns/wsdl/Data"
    location="http://specs.onename.com/specs/xns/gen/xns/Data.xsd"/>

<wsdl:message name="AddRequest">
  <wsdl:part name="body" type="msgs:AddRequest"/>
</wsdl:message>
<wsdl:message name="AddResponse">
  <wsdl:part name="body" type="msgs:AddResponse"/>
</wsdl:message>
<wsdl:message name="ChangeNotifyRequest">
  <wsdl:part name="body" type="msgs:ChangeNotifyRequest"/>
</wsdl:message>
<wsdl:message name="ChangeNotifyResponse">
  <wsdl:part name="body" type="msgs:ChangeNotifyResponse"/>
</wsdl:message>
<wsdl:message name="DeleteRequest">
  <wsdl:part name="body" type="msgs:DeleteRequest"/>
</wsdl:message>
<wsdl:message name="DeleteResponse">
  <wsdl:part name="body" type="msgs:DeleteResponse"/>
</wsdl:message>
<wsdl:message name="GetRequest">
  <wsdl:part name="body" type="msgs:GetRequest"/>
</wsdl:message>
<wsdl:message name="GetResponse">
  <wsdl:part name="body" type="msgs:GetResponse"/>
</wsdl:message>
<wsdl:message name="GetListElementsRequest">
  <wsdl:part name="body" type="msgs:GetListElementsRequest"/>
</wsdl:message>
<wsdl:message name="GetListElementsResponse">
  <wsdl:part name="body" type="msgs:GetListElementsResponse"/>
</wsdl:message>
<wsdl:message name="GetVersionsRequest">
  <wsdl:part name="body" type="msgs:GetVersionsRequest"/>
</wsdl:message>
<wsdl:message name="GetVersionsResponse">
  <wsdl:part name="body" type="msgs:GetVersionsResponse"/>
</wsdl:message>
<wsdl:message name="IncrementRequest">
```

```
  <wsdl:part name="body" type="msgs:IncrementRequest"/>
</wsdl:message>
<wsdl:message name="IncrementResponse">
  <wsdl:part name="body" type="msgs:IncrementResponse"/>
</wsdl:message>
<wsdl:message name="RespondToUpdateRequest">
  <wsdl:part name="body" type="msgs:RespondToUpdateRequest"/>
</wsdl:message>
<wsdl:message name="RespondToUpdateResponse">
  <wsdl:part name="body" type="msgs:RespondToUpdateResponse"/>
</wsdl:message>
<wsdl:message name="SetRequest">
  <wsdl:part name="body" type="msgs:SetRequest"/>
</wsdl:message>
<wsdl:message name="SetResponse">
  <wsdl:part name="body" type="msgs:SetResponse"/>
</wsdl:message>
<wsdl:message name="UpdateRequest">
  <wsdl:part name="body" type="msgs:UpdateRequest"/>
</wsdl:message>
<wsdl:message name="UpdateResponse">
  <wsdl:part name="body" type="msgs:UpdateResponse"/>
</wsdl:message>
<wsdl:message name="UpdateObjectRequest">
  <wsdl:part name="body" type="msgs:UpdateObjectRequest"/>
</wsdl:message>
<wsdl:message name="UpdateObjectResponse">
  <wsdl:part name="body" type="msgs:UpdateObjectResponse"/>
</wsdl:message>
<wsdl:portType name="Data">
  <wsdl:operation name="Add">
    <wsdl:input message="tns:AddRequest"/>
    <wsdl:output message="tns:AddResponse"/>
  </wsdl:operation>
  <wsdl:operation name="ChangeNotify">
    <wsdl:input message="tns:ChangeNotifyRequest"/>
    <wsdl:output message="tns:ChangeNotifyResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Delete">
    <wsdl:input message="tns:DeleteRequest"/>
    <wsdl:output message="tns:DeleteResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Get">
    <wsdl:input message="tns:GetRequest"/>
    <wsdl:output message="tns:GetResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetListElements">
    <wsdl:input message="tns:GetListElementsRequest"/>
    <wsdl:output message="tns:GetListElementsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetVersions">
    <wsdl:input message="tns:GetVersionsRequest"/>
    <wsdl:output message="tns:GetVersionsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Increment">
    <wsdl:input message="tns:IncrementRequest"/>
```

```
    <wsdl:output message="tns:IncrementResponse"/>
  </wsdl:operation>
  <wsdl:operation name="RespondToUpdate">
    <wsdl:input message="tns:RespondToUpdateRequest"/>
    <wsdl:output message="tns:RespondToUpdateResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Set">
    <wsdl:input message="tns:SetRequest"/>
    <wsdl:output message="tns:SetResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Update">
    <wsdl:input message="tns:UpdateRequest"/>
    <wsdl:output message="tns:UpdateResponse"/>
  </wsdl:operation>
  <wsdl:operation name="UpdateObject">
    <wsdl:input message="tns:UpdateObjectRequest"/>
    <wsdl:output message="tns:UpdateObjectResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Data" type="Data">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="Add">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:AddRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
    </wsdl:input>
    <wsdl:output message="tns:AddResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="ChangeNotify">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:ChangeNotifyRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
    </wsdl:input>
    <wsdl:output message="tns:ChangeNotifyResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Delete">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:DeleteRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
    </wsdl:input>
    <wsdl:output message="tns:DeleteResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Get">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GetRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
    </wsdl:input>
    <wsdl:output message="tns:GetResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
```

```
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetListElements">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:GetListElementsRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:input>
  <wsdl:output message="tns:GetListElementsResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetVersions">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:GetVersionsRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:input>
  <wsdl:output message="tns:GetVersionsResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Increment">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:IncrementRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:input>
  <wsdl:output message="tns:IncrementResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="RespondToUpdate">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:RespondToUpdateRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:input>
  <wsdl:output message="tns:RespondToUpdateResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Set">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:SetRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:input>
  <wsdl:output message="tns:SetResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Update">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:UpdateRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:input>
  <wsdl:output message="tns:UpdateResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
  </wsdl:output>
</wsdl:operation>
```

```
  <wsdl:operation name="UpdateObject">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:UpdateObjectRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
    </wsdl:input>
    <wsdl:output message="tns:UpdateObjectResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Data"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Service: Discovery

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="Discovery"
  targetNamespace="xns:+xns/wsdl/Discovery"
  xmlns:tns="xns:+xns/wsdl/Discovery"
  xmlns:msgs="xns:+xns/Discovery"
  xmlns:core="xns:+xns/Core"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    WSDL abstract service definition for the Discovery XNS service.
  </wsdl:documentation>
  <wsdl:import namespace="xns:+xns/Core"
    location="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
  <wsdl:import namespace="xns:+xns/wsdl/Discovery"
    location="http://specs.onename.com/specs/xns/gen/xns/Discovery.xsd"/>

<wsdl:message name="DescribeServiceRequest">
  <wsdl:part name="body" type="msgs:DescribeServiceRequest"/>
</wsdl:message>
<wsdl:message name="DescribeServiceResponse">
  <wsdl:part name="body" type="msgs:DescribeServiceResponse"/>
</wsdl:message>
<wsdl:message name="GetServiceProvidersRequest">
  <wsdl:part name="body" type="msgs:GetServiceProvidersRequest"/>
</wsdl:message>
<wsdl:message name="GetServiceProvidersResponse">
  <wsdl:part name="body" type="msgs:GetServiceProvidersResponse"/>
</wsdl:message>
<wsdl:message name="GetServicesDefinedRequest">
  <wsdl:part name="body" type="msgs:GetServicesDefinedRequest"/>
</wsdl:message>
<wsdl:message name="GetServicesDefinedResponse">
  <wsdl:part name="body" type="msgs:GetServicesDefinedResponse"/>
</wsdl:message>
<wsdl:message name="GetServicesOfferedRequest">
  <wsdl:part name="body" type="msgs:GetServicesOfferedRequest"/>
</wsdl:message>
<wsdl:message name="GetServicesOfferedResponse">
  <wsdl:part name="body" type="msgs:GetServicesOfferedResponse"/>
</wsdl:message>
<wsdl:message name="RegisterServiceProviderRequest">
  <wsdl:part name="body" type="msgs:RegisterServiceProviderRequest"/>
</wsdl:message>
<wsdl:message name="RegisterServiceProviderResponse">
  <wsdl:part name="body" type="msgs:RegisterServiceProviderResponse"/>
</wsdl:message>
<wsdl:message name="RetireServiceProviderRequest">
  <wsdl:part name="body" type="msgs:RetireServiceProviderRequest"/>
</wsdl:message>
<wsdl:message name="RetireServiceProviderResponse">
  <wsdl:part name="body" type="msgs:RetireServiceProviderResponse"/>
</wsdl:message>
<wsdl:portType name="Discovery">
```

```
  <wsdl:operation name="DescribeService">
    <wsdl:input message="tns:DescribeServiceRequest"/>
    <wsdl:output message="tns:DescribeServiceResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetServiceProviders">
    <wsdl:input message="tns:GetServiceProvidersRequest"/>
    <wsdl:output message="tns:GetServiceProvidersResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetServicesDefined">
    <wsdl:input message="tns:GetServicesDefinedRequest"/>
    <wsdl:output message="tns:GetServicesDefinedResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetServicesOffered">
    <wsdl:input message="tns:GetServicesOfferedRequest"/>
    <wsdl:output message="tns:GetServicesOfferedResponse"/>
  </wsdl:operation>
  <wsdl:operation name="RegisterServiceProvider">
    <wsdl:input message="tns:RegisterServiceProviderRequest"/>
    <wsdl:output message="tns:RegisterServiceProviderResponse"/>
  </wsdl:operation>
  <wsdl:operation name="RetireServiceProvider">
    <wsdl:input message="tns:RetireServiceProviderRequest"/>
    <wsdl:output message="tns:RetireServiceProviderResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Discovery" type="Discovery">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="DescribeService">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:DescribeServiceRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:input>
    <wsdl:output message="tns:DescribeServiceResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetServiceProviders">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GetServiceProvidersRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:input>
    <wsdl:output message="tns:GetServiceProvidersResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetServicesDefined">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GetServicesDefinedRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:input>
    <wsdl:output message="tns:GetServicesDefinedResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetServicesOffered">
```

```
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GetServicesOfferedRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:input>
    <wsdl:output message="tns:GetServicesOfferedResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="RegisterServiceProvider">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RegisterServiceProviderRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:input>
    <wsdl:output message="tns:RegisterServiceProviderResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="RetireServiceProvider">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RetireServiceProviderRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:input>
    <wsdl:output message="tns:RetireServiceProviderResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Discovery"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Service: Folder

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="Folder"
  targetNamespace="xns:+xns/wsdl/Folder"
  xmlns:tns="xns:+xns/wsdl/Folder"
  xmlns:msgs="xns:+xns/Folder"
  xmlns:core="xns:+xns/Core"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    WSDL abstract service definition for the Folder XNS service.
  </wsdl:documentation>
  <wsdl:import namespace="xns:+xns/Core"
    location="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
  <wsdl:import namespace="xns:+xns/wsdl/Folder"
    location="http://specs.onename.com/specs/xns/gen/xns/Folder.xsd"/>

<wsdl:message name="AddItemRequest">
  <wsdl:part name="body" type="msgs:AddItemRequest"/>
</wsdl:message>
<wsdl:message name="AddItemResponse">
  <wsdl:part name="body" type="msgs:AddItemResponse"/>
</wsdl:message>
<wsdl:message name="CreateRequest">
  <wsdl:part name="body" type="msgs:CreateRequest"/>
</wsdl:message>
<wsdl:message name="CreateResponse">
  <wsdl:part name="body" type="msgs:CreateResponse"/>
</wsdl:message>
<wsdl:message name="DeleteRequest">
  <wsdl:part name="body" type="msgs:DeleteRequest"/>
</wsdl:message>
<wsdl:message name="DeleteResponse">
  <wsdl:part name="body" type="msgs:DeleteResponse"/>
</wsdl:message>
<wsdl:message name="RemoveItemRequest">
  <wsdl:part name="body" type="msgs:RemoveItemRequest"/>
</wsdl:message>
<wsdl:message name="RemoveItemResponse">
  <wsdl:part name="body" type="msgs:RemoveItemResponse"/>
</wsdl:message>
<wsdl:message name="RenameRequest">
  <wsdl:part name="body" type="msgs:RenameRequest"/>
</wsdl:message>
<wsdl:message name="RenameResponse">
  <wsdl:part name="body" type="msgs:RenameResponse"/>
</wsdl:message>
<wsdl:message name="RenameItemRequest">
  <wsdl:part name="body" type="msgs:RenameItemRequest"/>
</wsdl:message>
<wsdl:message name="RenameItemResponse">
  <wsdl:part name="body" type="msgs:RenameItemResponse"/>
</wsdl:message>
<wsdl:portType name="Folder">
```

```
  <wsdl:operation name="AddItem">
    <wsdl:input message="tns:AddItemRequest"/>
    <wsdl:output message="tns:AddItemResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Create">
    <wsdl:input message="tns:CreateRequest"/>
    <wsdl:output message="tns:CreateResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Delete">
    <wsdl:input message="tns:DeleteRequest"/>
    <wsdl:output message="tns:DeleteResponse"/>
  </wsdl:operation>
  <wsdl:operation name="RemoveItem">
    <wsdl:input message="tns:RemoveItemRequest"/>
    <wsdl:output message="tns:RemoveItemResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Rename">
    <wsdl:input message="tns:RenameRequest"/>
    <wsdl:output message="tns:RenameResponse"/>
  </wsdl:operation>
  <wsdl:operation name="RenameItem">
    <wsdl:input message="tns:RenameItemRequest"/>
    <wsdl:output message="tns:RenameItemResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Folder" type="Folder">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="AddItem">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:AddItemRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:input>
    <wsdl:output message="tns:AddItemResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Create">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:CreateRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:input>
    <wsdl:output message="tns:CreateResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Delete">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:DeleteRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:input>
    <wsdl:output message="tns:DeleteResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="RemoveItem">
```

```
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RemoveItemRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:input>
    <wsdl:output message="tns:RemoveItemResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Rename">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RenameRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:input>
    <wsdl:output message="tns:RenameResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="RenameItem">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RenameItemRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:input>
    <wsdl:output message="tns:RenameItemResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Folder"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Service: Hosting

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="Hosting"
  targetNamespace="xns:+xns/wsdl/Hosting"
  xmlns:tns="xns:+xns/wsdl/Hosting"
  xmlns:msgs="xns:+xns/Hosting"
  xmlns:core="xns:+xns/Core"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    WSDL abstract service definition for the Hosting XNS service.
  </wsdl:documentation>
  <wsdl:import namespace="xns:+xns/Core"
    location="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
  <wsdl:import namespace="xns:+xns/wsdl/Hosting"
    location="http://specs.onename.com/specs/xns/gen/xns/Hosting.xsd"/>

<wsdl:message name="DeleteIdentityRequest">
  <wsdl:part name="body" type="msgs:DeleteIdentityRequest"/>
</wsdl:message>
<wsdl:message name="DeleteIdentityResponse">
  <wsdl:part name="body" type="msgs:DeleteIdentityResponse"/>
</wsdl:message>
<wsdl:message name="HostIdentityRequest">
  <wsdl:part name="body" type="msgs:HostIdentityRequest"/>
</wsdl:message>
<wsdl:message name="HostIdentityResponse">
  <wsdl:part name="body" type="msgs:HostIdentityResponse"/>
</wsdl:message>
<wsdl:portType name="Hosting">
  <wsdl:operation name="DeleteIdentity">
    <wsdl:input message="tns:DeleteIdentityRequest"/>
    <wsdl:output message="tns:DeleteIdentityResponse"/>
  </wsdl:operation>
  <wsdl:operation name="HostIdentity">
    <wsdl:input message="tns:HostIdentityRequest"/>
    <wsdl:output message="tns:HostIdentityResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Hosting" type="Hosting">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="DeleteIdentity">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:DeleteIdentityRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Hosting"/>
    </wsdl:input>
    <wsdl:output message="tns:DeleteIdentityResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Hosting"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="HostIdentity">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:HostIdentityRequest">
```

```
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Hosting"/>
    </wsdl:input>
    <wsdl:output message="tns:HostIdentityResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Hosting"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Service: ID

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="ID"
  targetNamespace="xns:+xns/wsdl/ID"
  xmlns:tns="xns:+xns/wsdl/ID"
  xmlns:msgs="xns:+xns/ID"
  xmlns:core="xns:+xns/Core"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    WSDL abstract service definition for the ID XNS service.
  </wsdl:documentation>
  <wsdl:import namespace="xns:+xns/Core"
    location="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
  <wsdl:import namespace="xns:+xns/wsdl/ID"
    location="http://specs.onename.com/specs/xns/gen/xns/ID.xsd"/>

<wsdl:message name="MapIDRequest">
  <wsdl:part name="body" type="msgs:MapIDRequest"/>
</wsdl:message>
<wsdl:message name="MapIDResponse">
  <wsdl:part name="body" type="msgs:MapIDResponse"/>
</wsdl:message>
<wsdl:message name="RegisterRequest">
  <wsdl:part name="body" type="msgs:RegisterRequest"/>
</wsdl:message>
<wsdl:message name="RegisterResponse">
  <wsdl:part name="body" type="msgs:RegisterResponse"/>
</wsdl:message>
<wsdl:message name="ResolveRequest">
  <wsdl:part name="body" type="msgs:ResolveRequest"/>
</wsdl:message>
<wsdl:message name="ResolveResponse">
  <wsdl:part name="body" type="msgs:ResolveResponse"/>
</wsdl:message>
<wsdl:message name="RetireRequest">
  <wsdl:part name="body" type="msgs:RetireRequest"/>
</wsdl:message>
<wsdl:message name="RetireResponse">
  <wsdl:part name="body" type="msgs:RetireResponse"/>
</wsdl:message>
<wsdl:portType name="ID">
  <wsdl:operation name="MapID">
    <wsdl:input message="tns:MapIDRequest"/>
    <wsdl:output message="tns:MapIDResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Register">
    <wsdl:input message="tns:RegisterRequest"/>
    <wsdl:output message="tns:RegisterResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Resolve">
    <wsdl:input message="tns:ResolveRequest"/>
    <wsdl:output message="tns:ResolveResponse"/>
  </wsdl:operation>
```

```
  <wsdl:operation name="Retire">
    <wsdl:input message="tns:RetireRequest"/>
    <wsdl:output message="tns:RetireResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="ID" type="ID">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="MapID">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:MapIDRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/ID"/>
    </wsdl:input>
    <wsdl:output message="tns:MapIDResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/ID"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Register">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RegisterRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/ID"/>
    </wsdl:input>
    <wsdl:output message="tns:RegisterResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/ID"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Resolve">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:ResolveRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/ID"/>
    </wsdl:input>
    <wsdl:output message="tns:ResolveResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/ID"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Retire">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RetireRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/ID"/>
    </wsdl:input>
    <wsdl:output message="tns:RetireResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/ID"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Service: Name

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="Name"
  targetNamespace="xns:+xns/wsdl/Name"
  xmlns:tns="xns:+xns/wsdl/Name"
  xmlns:msgs="xns:+xns/Name"
  xmlns:core="xns:+xns/Core"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    WSDL abstract service definition for the Name XNS service.
  </wsdl:documentation>
  <wsdl:import namespace="xns:+xns/Core"
    location="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
  <wsdl:import namespace="xns:+xns/wsdl/Name"
    location="http://specs.onename.com/specs/xns/gen/xns/Name.xsd"/>

<wsdl:message name="CloseNamespaceRequest">
  <wsdl:part name="body" type="msgs:CloseNamespaceRequest"/>
</wsdl:message>
<wsdl:message name="CloseNamespaceResponse">
  <wsdl:part name="body" type="msgs:CloseNamespaceResponse"/>
</wsdl:message>
<wsdl:message name="ExtendRequest">
  <wsdl:part name="body" type="msgs:ExtendRequest"/>
</wsdl:message>
<wsdl:message name="ExtendResponse">
  <wsdl:part name="body" type="msgs:ExtendResponse"/>
</wsdl:message>
<wsdl:message name="GetValidNamespacesRequest">
  <wsdl:part name="body" type="msgs:GetValidNamespacesRequest"/>
</wsdl:message>
<wsdl:message name="GetValidNamespacesResponse">
  <wsdl:part name="body" type="msgs:GetValidNamespacesResponse"/>
</wsdl:message>
<wsdl:message name="OpenNamespaceRequest">
  <wsdl:part name="body" type="msgs:OpenNamespaceRequest"/>
</wsdl:message>
<wsdl:message name="OpenNamespaceResponse">
  <wsdl:part name="body" type="msgs:OpenNamespaceResponse"/>
</wsdl:message>
<wsdl:message name="RegisterRequest">
  <wsdl:part name="body" type="msgs:RegisterRequest"/>
</wsdl:message>
<wsdl:message name="RegisterResponse">
  <wsdl:part name="body" type="msgs:RegisterResponse"/>
</wsdl:message>
<wsdl:message name="ReleaseRequest">
  <wsdl:part name="body" type="msgs:ReleaseRequest"/>
</wsdl:message>
<wsdl:message name="ReleaseResponse">
  <wsdl:part name="body" type="msgs:ReleaseResponse"/>
</wsdl:message>
<wsdl:message name="RenameRequest">
```

```
  <wsdl:part name="body" type="msgs:RenameRequest"/>
</wsdl:message>
<wsdl:message name="RenameResponse">
  <wsdl:part name="body" type="msgs:RenameResponse"/>
</wsdl:message>
<wsdl:message name="RequestRequest">
  <wsdl:part name="body" type="msgs:RequestRequest"/>
</wsdl:message>
<wsdl:message name="RequestResponse">
  <wsdl:part name="body" type="msgs:RequestResponse"/>
</wsdl:message>
<wsdl:message name="ResolveRequest">
  <wsdl:part name="body" type="msgs:ResolveRequest"/>
</wsdl:message>
<wsdl:message name="ResolveResponse">
  <wsdl:part name="body" type="msgs:ResolveResponse"/>
</wsdl:message>
<wsdl:message name="ResolveNamespaceRequest">
  <wsdl:part name="body" type="msgs:ResolveNamespaceRequest"/>
</wsdl:message>
<wsdl:message name="ResolveNamespaceResponse">
  <wsdl:part name="body" type="msgs:ResolveNamespaceResponse"/>
</wsdl:message>
<wsdl:message name="TransferRequest">
  <wsdl:part name="body" type="msgs:TransferRequest"/>
</wsdl:message>
<wsdl:message name="TransferResponse">
  <wsdl:part name="body" type="msgs:TransferResponse"/>
</wsdl:message>
<wsdl:portType name="Name">
  <wsdl:operation name="CloseNamespace">
    <wsdl:input message="tns:CloseNamespaceRequest"/>
    <wsdl:output message="tns:CloseNamespaceResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Extend">
    <wsdl:input message="tns:ExtendRequest"/>
    <wsdl:output message="tns:ExtendResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetValidNamespaces">
    <wsdl:input message="tns:GetValidNamespacesRequest"/>
    <wsdl:output message="tns:GetValidNamespacesResponse"/>
  </wsdl:operation>
  <wsdl:operation name="OpenNamespace">
    <wsdl:input message="tns:OpenNamespaceRequest"/>
    <wsdl:output message="tns:OpenNamespaceResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Register">
    <wsdl:input message="tns:RegisterRequest"/>
    <wsdl:output message="tns:RegisterResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Release">
    <wsdl:input message="tns:ReleaseRequest"/>
    <wsdl:output message="tns:ReleaseResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Rename">
    <wsdl:input message="tns:RenameRequest"/>
```

```
    <wsdl:output message="tns:RenameResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Request">
    <wsdl:input message="tns:RequestRequest"/>
    <wsdl:output message="tns:RequestResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Resolve">
    <wsdl:input message="tns:ResolveRequest"/>
    <wsdl:output message="tns:ResolveResponse"/>
  </wsdl:operation>
  <wsdl:operation name="ResolveNamespace">
    <wsdl:input message="tns:ResolveNamespaceRequest"/>
    <wsdl:output message="tns:ResolveNamespaceResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Transfer">
    <wsdl:input message="tns:TransferRequest"/>
    <wsdl:output message="tns:TransferResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Name" type="Name">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="CloseNamespace">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:CloseNamespaceRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:CloseNamespaceResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Extend">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:ExtendRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:ExtendResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetValidNamespaces">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GetValidNamespacesRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:GetValidNamespacesResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="OpenNamespace">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:OpenNamespaceRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:OpenNamespaceResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
```

```
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Register">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RegisterRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:RegisterResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Release">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:ReleaseRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:ReleaseResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Rename">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RenameRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:RenameResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Request">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:RequestRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:RequestResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Resolve">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:ResolveRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:ResolveResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="ResolveNamespace">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:ResolveNamespaceRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:ResolveNamespaceResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:output>
  </wsdl:operation>
```

```
  <wsdl:operation name="Transfer">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:TransferRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:input>
    <wsdl:output message="tns:TransferResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Name"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Service: Negotiation

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="Negotiation"
  targetNamespace="xns:+xns/wsdl/Negotiation"
  xmlns:tns="xns:+xns/wsdl/Negotiation"
  xmlns:msgs="xns:+xns/Negotiation"
  xmlns:core="xns:+xns/Core"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    WSDL abstract service definition for the Negotiation XNS service.
  </wsdl:documentation>
  <wsdl:import namespace="xns:+xns/Core"
    location="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
  <wsdl:import namespace="xns:+xns/wsdl/Negotiation"
    location="http://specs.onename.com/specs/xns/gen/xns/Negotiation.xsd"/>

<wsdl:message name="ConfirmNegotiationRequest">
  <wsdl:part name="body" type="msgs:ConfirmNegotiationRequest"/>
</wsdl:message>
<wsdl:message name="ConfirmNegotiationResponse">
  <wsdl:part name="body" type="msgs:ConfirmNegotiationResponse"/>
</wsdl:message>
<wsdl:message name="ConfirmReceiptAcceptedRequest">
  <wsdl:part name="body" type="msgs:ConfirmReceiptAcceptedRequest"/>
</wsdl:message>
<wsdl:message name="ConfirmReceiptAcceptedResponse">
  <wsdl:part name="body" type="msgs:ConfirmReceiptAcceptedResponse"/>
</wsdl:message>
<wsdl:message name="NegotiateContractRequest">
  <wsdl:part name="body" type="msgs:NegotiateContractRequest"/>
</wsdl:message>
<wsdl:message name="NegotiateContractResponse">
  <wsdl:part name="body" type="msgs:NegotiateContractResponse"/>
</wsdl:message>
<wsdl:message name="RequestFormRequest">
  <wsdl:part name="body" type="msgs:RequestFormRequest"/>
</wsdl:message>
<wsdl:message name="RequestFormResponse">
  <wsdl:part name="body" type="msgs:RequestFormResponse"/>
</wsdl:message>
<wsdl:message name="SubmitForConfirmationRequest">
  <wsdl:part name="body" type="msgs:SubmitForConfirmationRequest"/>
</wsdl:message>
<wsdl:message name="SubmitForConfirmationResponse">
  <wsdl:part name="body" type="msgs:SubmitForConfirmationResponse"/>
</wsdl:message>
<wsdl:message name="SubmitForNegotiationRequest">
  <wsdl:part name="body" type="msgs:SubmitForNegotiationRequest"/>
</wsdl:message>
<wsdl:message name="SubmitForNegotiationResponse">
  <wsdl:part name="body" type="msgs:SubmitForNegotiationResponse"/>
</wsdl:message>
<wsdl:message name="SubmitReceiptRequest">
```

```
  <wsdl:part name="body" type="msgs:SubmitReceiptRequest"/>
</wsdl:message>
<wsdl:message name="SubmitReceiptResponse">
  <wsdl:part name="body" type="msgs:SubmitReceiptResponse"/>
</wsdl:message>
<wsdl:message name="TerminateContractRequest">
  <wsdl:part name="body" type="msgs:TerminateContractRequest"/>
</wsdl:message>
<wsdl:message name="TerminateContractResponse">
  <wsdl:part name="body" type="msgs:TerminateContractResponse"/>
</wsdl:message>
<wsdl:portType name="Negotiation">
  <wsdl:operation name="ConfirmNegotiation">
    <wsdl:input message="tns:ConfirmNegotiationRequest"/>
    <wsdl:output message="tns:ConfirmNegotiationResponse"/>
  </wsdl:operation>
  <wsdl:operation name="ConfirmReceiptAccepted">
    <wsdl:input message="tns:ConfirmReceiptAcceptedRequest"/>
    <wsdl:output message="tns:ConfirmReceiptAcceptedResponse"/>
  </wsdl:operation>
  <wsdl:operation name="NegotiateContract">
    <wsdl:input message="tns:NegotiateContractRequest"/>
    <wsdl:output message="tns:NegotiateContractResponse"/>
  </wsdl:operation>
  <wsdl:operation name="RequestForm">
    <wsdl:input message="tns:RequestFormRequest"/>
    <wsdl:output message="tns:RequestFormResponse"/>
  </wsdl:operation>
  <wsdl:operation name="SubmitForConfirmation">
    <wsdl:input message="tns:SubmitForConfirmationRequest"/>
    <wsdl:output message="tns:SubmitForConfirmationResponse"/>
  </wsdl:operation>
  <wsdl:operation name="SubmitForNegotiation">
    <wsdl:input message="tns:SubmitForNegotiationRequest"/>
    <wsdl:output message="tns:SubmitForNegotiationResponse"/>
  </wsdl:operation>
  <wsdl:operation name="SubmitReceipt">
    <wsdl:input message="tns:SubmitReceiptRequest"/>
    <wsdl:output message="tns:SubmitReceiptResponse"/>
  </wsdl:operation>
  <wsdl:operation name="TerminateContract">
    <wsdl:input message="tns:TerminateContractRequest"/>
    <wsdl:output message="tns:TerminateContractResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="Negotiation" type="Negotiation">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="ConfirmNegotiation">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:ConfirmNegotiationRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
    </wsdl:input>
    <wsdl:output message="tns:ConfirmNegotiationResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
    </wsdl:output>
  </wsdl:operation>
```

```
<wsdl:operation name="ConfirmReceiptAccepted">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:ConfirmReceiptAcceptedRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:input>
  <wsdl:output message="tns:ConfirmReceiptAcceptedResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="NegotiateContract">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:NegotiateContractRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:input>
  <wsdl:output message="tns:NegotiateContractResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="RequestForm">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:RequestFormRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:input>
  <wsdl:output message="tns:RequestFormResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="SubmitForConfirmation">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:SubmitForConfirmationRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:input>
  <wsdl:output message="tns:SubmitForConfirmationResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="SubmitForNegotiation">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:SubmitForNegotiationRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:input>
  <wsdl:output message="tns:SubmitForNegotiationResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="SubmitReceipt">
  <soap:operation soapAction="" style="document"/>
  <wsdl:input message="tns:SubmitReceiptRequest">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:input>
  <wsdl:output message="tns:SubmitReceiptResponse">
    <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="TerminateContract">
  <soap:operation soapAction="" style="document"/>
```

```
    <wsdl:input message="tns:TerminateContractRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
    </wsdl:input>
    <wsdl:output message="tns:TerminateContractResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Negotiation"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Service: Session

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="Session"
  targetNamespace="xns:+xns/wsdl/Session"
  xmlns:tns="xns:+xns/wsdl/Session"
  xmlns:msgs="xns:+xns/Session"
  xmlns:core="xns:+xns/Core"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    WSDL abstract service definition for the Session XNS service.
  </wsdl:documentation>
  <wsdl:import namespace="xns:+xns/Core"
    location="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
  <wsdl:import namespace="xns:+xns/wsdl/Session"
    location="http://specs.onename.com/specs/xns/gen/xns/Session.xsd"/>

<wsdl:message name="AuthenticateRequest">
  <wsdl:part name="body" type="msgs:AuthenticateRequest"/>
</wsdl:message>
<wsdl:message name="AuthenticateResponse">
  <wsdl:part name="body" type="msgs:AuthenticateResponse"/>
</wsdl:message>
<wsdl:message name="GetSessionURIRequest">
  <wsdl:part name="body" type="msgs:GetSessionURIRequest"/>
</wsdl:message>
<wsdl:message name="GetSessionURIResponse">
  <wsdl:part name="body" type="msgs:GetSessionURIResponse"/>
</wsdl:message>
<wsdl:message name="LoginRequest">
  <wsdl:part name="body" type="msgs:LoginRequest"/>
</wsdl:message>
<wsdl:message name="LoginResponse">
  <wsdl:part name="body" type="msgs:LoginResponse"/>
</wsdl:message>
<wsdl:message name="LoginNotifyRequest">
  <wsdl:part name="body" type="msgs:LoginNotifyRequest"/>
</wsdl:message>
<wsdl:message name="LoginNotifyResponse">
  <wsdl:part name="body" type="msgs:LoginNotifyResponse"/>
</wsdl:message>
<wsdl:message name="LogoutNotifyRequest">
  <wsdl:part name="body" type="msgs:LogoutNotifyRequest"/>
</wsdl:message>
<wsdl:message name="LogoutNotifyResponse">
  <wsdl:part name="body" type="msgs:LogoutNotifyResponse"/>
</wsdl:message>
<wsdl:message name="SubmitAuthCertRequest">
  <wsdl:part name="body" type="msgs:SubmitAuthCertRequest"/>
</wsdl:message>
<wsdl:message name="SubmitAuthCertResponse">
  <wsdl:part name="body" type="msgs:SubmitAuthCertResponse"/>
</wsdl:message>
<wsdl:message name="XNSSessionLogoutRequest">
```

```
  <wsdl:part name="body" type="msgs:XNSSessionLogoutRequest"/>
</wsdl:message>
<wsdl:message name="XNSSessionLogoutResponse">
  <wsdl:part name="body" type="msgs:XNSSessionLogoutResponse"/>
</wsdl:message>
<wsdl:portType name="Session">
  <wsdl:operation name="Authenticate">
    <wsdl:input message="tns:AuthenticateRequest"/>
    <wsdl:output message="tns:AuthenticateResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetSessionURI">
    <wsdl:input message="tns:GetSessionURIRequest"/>
    <wsdl:output message="tns:GetSessionURIResponse"/>
  </wsdl:operation>
  <wsdl:operation name="Login">
    <wsdl:input message="tns:LoginRequest"/>
    <wsdl:output message="tns:LoginResponse"/>
  </wsdl:operation>
  <wsdl:operation name="LoginNotify">
    <wsdl:input message="tns:LoginNotifyRequest"/>
    <wsdl:output message="tns:LoginNotifyResponse"/>
  </wsdl:operation>
  <wsdl:operation name="LogoutNotify">
    <wsdl:input message="tns:LogoutNotifyRequest"/>
    <wsdl:output message="tns:LogoutNotifyResponse"/>
  </wsdl:operation>
  <wsdl:operation name="SubmitAuthCert">
    <wsdl:input message="tns:SubmitAuthCertRequest"/>
    <wsdl:output message="tns:SubmitAuthCertResponse"/>
  </wsdl:operation>
  <wsdl:operation name="XNSSessionLogout">
    <wsdl:input message="tns:XNSSessionLogoutRequest"/>
    <wsdl:output message="tns:XNSSessionLogoutResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Session" type="Session">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="Authenticate">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:AuthenticateRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:input>
    <wsdl:output message="tns:AuthenticateResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetSessionURI">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:GetSessionURIRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:input>
    <wsdl:output message="tns:GetSessionURIResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:output>
  </wsdl:operation>
```

```
  <wsdl:operation name="Login">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:LoginRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:input>
    <wsdl:output message="tns:LoginResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="LoginNotify">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:LoginNotifyRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:input>
    <wsdl:output message="tns:LoginNotifyResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="LogoutNotify">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:LogoutNotifyRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:input>
    <wsdl:output message="tns:LogoutNotifyResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="SubmitAuthCert">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:SubmitAuthCertRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:input>
    <wsdl:output message="tns:SubmitAuthCertResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="XNSSessionLogout">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input message="tns:XNSSessionLogoutRequest">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:input>
    <wsdl:output message="tns:XNSSessionLogoutResponse">
      <soap:body use="literal" parts="body" namespace="xns:+xns/wsdl/Session"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

APPENDIX **B**

XSD Files (Normative)

This Appendix contains XML Schema Definition (XSD) files for each of the eleven XNS base services. These files are normative. Machine-readable versions of these files can be downloaded from www.xns.org.

Service: Core

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="Core" targetNamespace="xns:+xns/Core"
    xmlns:authentication="xns:+xns/Authentication"
    xmlns:certification="xns:+xns/Certification"
    xmlns:tns="xns:+xns/Core"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Authentication"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Authentication.xsd"/>
    <xsd:import namespace="xns:+xns/Certification"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Certification.xsd"/>
<xsd:simpleType name="XNSAddress">
        <xsd:union memberTypes="tns:XNSID tns:XNSName tns:AbsoluteAddress"/>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="XNSAddressList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="XNSAddress" type="tns:XNSAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="XNSID">
        <xsd:union memberTypes="tns:IdentityID tns:IdentityDataID tns:DataID
tns:RelativeDataID"/>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="XNSIDList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="XNSID" type="tns:XNSID"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="IdentityID">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="IdentityIDList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="IdentityID" type="tns:IdentityID"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="HostID">
```

```
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="HostIDList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="HostID" type="tns:HostID"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="OID">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\p{N}+ (. \p{N}+)*"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="OIDList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="OID" type="tns:OID"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="UUID">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-
f]{12}"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="UUIDList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="UUID" type="tns:UUID"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="IdentityIDNode">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="ID">
        <xsd:restriction base="xsd:Name">
            <xsd:pattern value="(^[@=+$/]^[/]*)+"/>
            <xsd:maxLength value="64"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="IDList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
```

```
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="ID" type="tns:ID"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="IdentityDataID">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:complexType mixed="true" name="IdentityDataIDList">
    <xsd:complexContent>
        <xsd:extension base="tns:List">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="IdentityDataID" type="tns:IdentityDataID"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="DataID">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:complexType mixed="true" name="DataIDList">
    <xsd:complexContent>
        <xsd:extension base="tns:List">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="DataID" type="tns:DataID"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="RelativeDataID">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:complexType mixed="true" name="RelativeDataIDList">
    <xsd:complexContent>
        <xsd:extension base="tns:List">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="RelativeDataID" type="tns:RelativeDataID"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="DataIDNode">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:simpleType name="Version">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value=",v \p{N}+ | ,t [0-9:-]+"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="VersionNumber">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\p{N}+"/>
```

```
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="VersionDate">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="[0-9:-]+"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="XNSName">
        <xsd:union memberTypes="tns:IdentityName tns:IdentityDataName tns:DataName
tns:RelativeDataName"/>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="XNSNameList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="XNSName" type="tns:XNSName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="IdentityName">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="IdentityNameList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="IdentityName" type="tns:IdentityName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="NamespaceSymbol">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="= | @ | +"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="IdentityNameNode">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="Name">
        <xsd:restriction base="xsd:Name">
            <xsd:pattern value="(^[@=+$/]^[/]*)+"/>
            <xsd:maxLength value="64"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="NameList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Name" type="tns:Name"/>
                </xsd:sequence>
            </xsd:extension>
```

```
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="IdentityAddress">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:complexType mixed="true" name="IdentityAddressList">
    <xsd:complexContent>
        <xsd:extension base="tns:List">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="IdentityAddress" type="tns:IdentityAddress"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="IdentityDataName">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:complexType mixed="true" name="IdentityDataNameList">
    <xsd:complexContent>
        <xsd:extension base="tns:List">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="IdentityDataName" type="tns:IdentityDataName"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="DataName">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:complexType mixed="true" name="DataNameList">
    <xsd:complexContent>
        <xsd:extension base="tns:List">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="DataName" type="tns:DataName"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="RelativeDataName">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:complexType mixed="true" name="RelativeDataNameList">
    <xsd:complexContent>
        <xsd:extension base="tns:List">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="RelativeDataName" type="tns:RelativeDataName"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="DataNameNode">
    <xsd:restriction base="xsd:string"/>
```

```
</xsd:simpleType>
<xsd:simpleType name="IdentityDataAddress">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
<xsd:complexType mixed="true" name="IdentityDataAddressList">
    <xsd:complexContent>
        <xsd:extension base="tns:List">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="IdentityDataAddress" type="tns:IdentityDataAddress"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="AbsoluteAddress">
    <xsd:union memberTypes="tns:IdentityAddress tns:IdentityDataAddress"/>
</xsd:simpleType>
<xsd:complexType mixed="true" name="AbsoluteAddressList">
    <xsd:complexContent>
        <xsd:extension base="tns:List">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="AbsoluteAddress" type="tns:AbsoluteAddress"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="true" mixed="true" name="ElementList">
    <xsd:attribute name="Name" type="tns:Name" use="optional"/>
    <xsd:attribute name="ID" type="tns:ID" use="required"/>
    <xsd:attribute default="false" name="PN" type="xsd:boolean" use="optional"/>
    <xsd:attribute fixed="true" name="PI" type="xsd:boolean"/>
    <xsd:attribute name="type" type="tns:IdentityDataAddress" use="optional"/>
</xsd:complexType>
<xsd:complexType mixed="true" name="XLink">
    <xsd:attribute name="Identity" type="tns:IdentityAddress" use="required"/>
    <xsd:attribute name="ID" type="tns:ID" use="optional"/>
    <xsd:attribute name="ExtID" type="tns:DataID" use="optional"/>
    <xsd:attribute name="LastUpdate" type="xsd:dateTime" use="optional"/>
    <xsd:attribute name="Contract" type="tns:IdentityDataAddress" use="optional"/>
</xsd:complexType>
<xsd:complexType mixed="true" name="XLinkList">
    <xsd:complexContent>
        <xsd:extension base="tns:List">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="XLink" type="tns:XLink"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="true" mixed="true" name="List">
    <xsd:attribute name="Name" type="tns:Name" use="optional"/>
    <xsd:attribute name="ID" type="tns:ID" use="required"/>
    <xsd:attribute fixed="true" name="PI" type="xsd:boolean"/>
</xsd:complexType>
```

```
    <xsd:complexType mixed="true" name="Message">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSObject"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MessageList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Message" type="tns:Message"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="TextList">
        <xsd:complexContent>
            <xsd:extension base="tns:List">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Text" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="XNSObject">
        <xsd:sequence>
            <xsd:element minOccurs="0" name="Description" type="xsd:string"/>
            <xsd:element minOccurs="0" name="Notes" type="xsd:string"/>
            <xsd:element minOccurs="0" name="XNSCerts"
type="certification:CertificateList"/>
            <xsd:element minOccurs="0" name="XNSBackRefs" type="tns:RefList"/>
            <xsd:element minOccurs="1" name="XNSCreateDate" type="xsd:dateTime"/>
            <xsd:element minOccurs="0" name="XNSUpdateDate" type="xsd:dateTime"/>
            <xsd:element minOccurs="0" name="XNSOnUpdate" type="tns:TextList"/>
            <xsd:element minOccurs="0" name="XNSVersions" type="tns:RefList"/>
            <xsd:element minOccurs="0" name="XNSLinks" type="tns:XLinkList"/>
            <xsd:element minOccurs="0" name="XNSXRefs" type="tns:XNSAddressList"/>
        </xsd:sequence>
        <xsd:attribute name="Name" type="tns:Name" use="optional"/>
        <xsd:attribute name="IName" type="tns:Name" use="optional"/>
        <xsd:attribute name="ID" type="tns:ID" use="required"/>
        <xsd:attribute name="IC" type="xsd:long" use="optional"/>
        <xsd:attribute default="false" name="PN" type="xsd:boolean" use="optional"/>
        <xsd:attribute default="false" name="PI" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="XA" type="tns:XNSAddress" use="optional"/>
        <xsd:attribute name="Ref" type="tns:XNSID" use="optional"/>
        <xsd:attribute name="SV" type="xsd:nonNegativeInteger" use="optional"/>
        <xsd:attribute name="IV" type="xsd:nonNegativeInteger" use="optional"/>
        <xsd:attribute default="false" name="AV" type="xsd:boolean" use="optional"/>
       <xsd:attribute default="false" name="Public" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="type" type="tns:IdentityDataAddress" use="optional"/>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="XNSObjectList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
```

```
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="XNSObject" type="tns:XNSObject"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Domain">
        <xsd:complexContent>
            <xsd:extension base="tns:Group">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="DomainOwner"
type="tns:DomainOwner"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DomainList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Domain" type="tns:Domain"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DomainOwner">
        <xsd:complexContent>
            <xsd:extension base="tns:GroupOwner">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Domain" type="tns:Domain"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DomainOwnerList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="DomainOwner" type="tns:DomainOwner"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Group">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="GroupOwner" type="tns:GroupOwner"/>
                    <xsd:element minOccurs="0" name="Groups"
type="tns:IdentityDataAddressList"/>
                    <xsd:element minOccurs="0" name="Members"
type="tns:IdentityAddressList"/>
```

```
                    <xsd:element minOccurs="0" name="Memberships"
type="tns:MembershipList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GroupList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Group" type="tns:Group"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GroupOwner">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Group" type="tns:Group"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GroupOwnerList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="GroupOwner" type="tns:GroupOwner"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="INT">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSObject"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="INTList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="INT" type="tns:INT"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="IdentityDef">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="IdentityType"
type="tns:IdentityTypeEnum"/>
```

```
                    <xsd:element minOccurs="0" name="Memberships"
type="tns:MembershipList"/>
                    <xsd:element minOccurs="0" name="PublicKey"
type="certification:PublicKeyCert"/>
                    <xsd:element minOccurs="0" name="Types" type="tns:XNSObjectList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="IdentityDefList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="IdentityDef" type="tns:IdentityDef"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="IdentityTypeEnum">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="PER"/>
            <xsd:enumeration value="ORG"/>
            <xsd:enumeration value="GEN"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="Membership">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Group"
type="tns:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="GroupOwner"
type="tns:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="GroupType"
type="tns:IdentityDataAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MembershipList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Membership" type="tns:Membership"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="MsgStateEnum">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="REQUEST"/>
            <xsd:enumeration value="RESPONSE"/>
            <xsd:enumeration value="EXCEPTION"/>
        </xsd:restriction>
```

```
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="Ref">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSObject"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RefList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Ref" type="tns:Ref"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="URN">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSObject"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="URNList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="URN" type="tns:URN"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="XNSException">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Description" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Name" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="XNSException"
type="tns:XNSException"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="XNSExceptionList">
        <xsd:complexContent>
            <xsd:extension base="tns:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="XNSException" type="tns:XNSException"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Service: Authentication

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="Authentication"
    targetNamespace="xns:+xns/Authentication"
    xmlns:certification="xns:+xns/Certification"
    xmlns:core="xns:+xns/Core" xmlns:tns="xns:+xns/Authentication"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Core"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
    <xsd:import namespace="xns:+xns/Certification"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Certification.xsd"/>
    <xsd:complexType mixed="true" name="CertifySession">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="AuthCert" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Compress" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="EncryptFor"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="SignBy"
type="core:IdentityAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CertifySessionRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:CertifySession"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CertifySessionResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:CertifySession"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ChangeCredentials">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="NewCredential"
type="tns:Credential"/>
                    <xsd:element minOccurs="0" name="OldCredential"
type="tns:Credential"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ChangeCredentialsRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:ChangeCredentials"/>
        </xsd:complexContent>
    </xsd:complexType>
```

```
    <xsd:complexType mixed="true" name="ChangeCredentialsResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:ChangeCredentials"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetWebLoginURI">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="WebLoginURI" type="xsd:anyURI"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetWebLoginURIRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:GetWebLoginURI"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetWebLoginURIResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:GetWebLoginURI"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Login">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="AuthSession"
type="tns:AuthSession"/>
                    <xsd:element minOccurs="0" name="Credentials"
type="tns:CredentialList"/>
                    <xsd:element minOccurs="0" name="IdentityAddress"
type="core:IdentityAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LoginRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Login"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LoginResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Login"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Logout">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="AuthSession"
type="core:IdentityDataAddress"/>
                </xsd:sequence>
            </xsd:extension>
```

```
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="LogoutRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:Logout"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="LogoutResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:Logout"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="AuthSession">
    <xsd:complexContent>
        <xsd:extension base="core:XNSObject">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="ConnectionID" type="xsd:string"/>
                <xsd:element minOccurs="0" name="SessionHandle" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="AuthSessionList">
    <xsd:complexContent>
        <xsd:extension base="core:ElementList">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="AuthSession" type="tns:AuthSession"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="Credential">
    <xsd:complexContent>
        <xsd:extension base="core:XNSObject"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="CredentialList">
    <xsd:complexContent>
        <xsd:extension base="core:ElementList">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="Credential" type="tns:Credential"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="HashAlgorithmEnum">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="MD5"/>
        <xsd:enumeration value="NONE"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType mixed="true" name="PasswordCredential">
    <xsd:complexContent>
        <xsd:extension base="tns:Credential">
```

```
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="HashAlgorithm"
type="tns:HashAlgorithmEnum"/>
                    <xsd:element minOccurs="0" name="Password" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="UserName" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="PasswordCredentialList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="PasswordCredential" type="tns:PasswordCredential"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="PersonalIdentityCredential">
        <xsd:complexContent>
            <xsd:extension base="tns:Credential">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="AuthCert" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Compressed" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="EncryptedFor"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="PersonalIdentity"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="SignedBy"
type="core:IdentityAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="PersonalIdentityCredentialList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="PersonalIdentityCredential"
type="tns:PersonalIdentityCredential"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Session">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="AuthCert"
type="certification:AuthenticationCert"/>
                    <xsd:element minOccurs="0" name="IdentityAddress"
type="core:IdentityAddress"/>
                </xsd:sequence>
            </xsd:extension>
```

```
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SessionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Session" type="tns:Session"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Service: Certification

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="Certification" targetNamespace="xns:+xns/Certification"
    xmlns:authentication="xns:+xns/Authentication"
    xmlns:core="xns:+xns/Core" xmlns:tns="xns:+xns/Certification"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Core"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
    <xsd:import namespace="xns:+xns/Authentication"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Authentication.xsd"/>
    <xsd:complexType mixed="true" name="Authorize">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Message" type="core:Message"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AuthorizeRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Authorize"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AuthorizeResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Authorize"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmAttrCert">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Cert" type="tns:AttributeCert"/>
                    <xsd:element minOccurs="0" name="Certified" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Exception"
type="core:XNSException"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmAttrCertRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:ConfirmAttrCert"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmAttrCertResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:ConfirmAttrCert"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmRevocation">
```

```
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="CertificateID"
type="core:IdentityDataID"/>
                    <xsd:element minOccurs="0" name="Reason" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmRevocationRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:ConfirmRevocation"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmRevocationResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:ConfirmRevocation"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GenerateKeyPair">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="PublicKey"
type="tns:PublicKeyCert"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GenerateKeyPairRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:GenerateKeyPair"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GenerateKeyPairResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:GenerateKeyPair"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetCACerts">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="CACerts" type="tns:CACertList"/>
                    <xsd:element minOccurs="0" name="CertType"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="DataType"
type="core:IdentityDataAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetCACertsRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:GetCACerts"/>
```

```
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetCACertsResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:GetCACerts"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetCRL">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="CAIdentity"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="CRL" type="tns:CRL"/>
                    <xsd:element minOccurs="0" name="CertType"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="DataType"
type="core:IdentityDataAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetCRLRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:GetCRL"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetCRLResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:GetCRL"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetPublicKeys">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Identity"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="PublicKeys"
type="tns:PublicKeyCertList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetPublicKeysRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:GetPublicKeys"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetPublicKeysResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:GetPublicKeys"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RequestAttrCert">
        <xsd:complexContent>
```

```
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Cert" type="tns:AttributeCert"/>
                    <xsd:element minOccurs="0" name="Pending" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="ToCertify" type="core:XNSObject"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RequestAttrCertRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:RequestAttrCert"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RequestAttrCertResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:RequestAttrCert"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RevokeCert">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="CertificateID"
type="core:IdentityDataID"/>
                    <xsd:element minOccurs="0" name="Reason" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RevokeCertRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:RevokeCert"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RevokeCertResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:RevokeCert"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitForAttrCertConfirm">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Cert" type="tns:AttributeCert"/>
                    <xsd:element minOccurs="0" name="Certified" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Exception"
type="core:XNSException"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitForAttrCertConfirmRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:SubmitForAttrCertConfirm"/>
        </xsd:complexContent>
```

```
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitForAttrCertConfirmResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:SubmitForAttrCertConfirm"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="VerifyCert">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="CertificateID"
type="core:IdentityDataID"/>
                    <xsd:element minOccurs="0"
                        name="NotifyUponRevocation" type="core:IdentityAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="VerifyCertRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:VerifyCert"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="VerifyCertResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:VerifyCert"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="VerifySignature">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Certificate"
type="tns:Certificate"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="VerifySignatureRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:VerifySignature"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="VerifySignatureResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:VerifySignature"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AttributeCert">
        <xsd:complexContent>
            <xsd:extension base="tns:Certificate">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Assertion"
type="tns:SAMLAttributeAssertion"/>
                    <xsd:element minOccurs="0" name="Certifying"
type="core:IdentityDataID"/>
```

```
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AttributeCertList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="AttributeCert" type="tns:AttributeCert"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AuthenticationCert">
        <xsd:complexContent>
            <xsd:extension base="tns:Certificate">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Assertion"
type="tns:SAMLAuthenticationAssertion"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AuthenticationCertList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="AuthenticationCert" type="tns:AuthenticationCert"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AuthorizationCert">
        <xsd:complexContent>
            <xsd:extension base="tns:Certificate">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Assertion"
type="tns:SAMLAuthorizationDecisionAssertion"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AuthorizationCertList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="AuthorizationCert" type="tns:AuthorizationCert"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CACert">
        <xsd:complexContent>
```

```
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="CertType"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="DataType"
type="core:IdentityDataAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CACertList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="CACert" type="tns:CACert"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CRL">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="CertType"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="DataType"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="Revoked" type="core:DataIDList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CRLList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="CRL" type="tns:CRL"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Certificate">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="NotifyOnRevocation"
type="core:IdentityAddressList"/>
                    <xsd:element minOccurs="0" name="RevocationDate"
type="xsd:dateTime"/>
                    <xsd:element minOccurs="0" name="RevocationReason"
type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
```

```
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CertificateList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Certificate" type="tns:Certificate"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CredentialCert">
        <xsd:complexContent>
            <xsd:extension base="tns:Certificate">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Credential"
type="authentication:Credential"/>
                   <xsd:element minOccurs="0" name="DateCollected" type="xsd:dateTime"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CredentialCertList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="CredentialCert" type="tns:CredentialCert"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="PublicKeyCert">
        <xsd:complexContent>
            <xsd:extension base="tns:Certificate">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Key" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="PublicKeyCertList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="PublicKeyCert" type="tns:PublicKeyCert"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAssertion">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Audience" type="xsd:anyURI"/>
```

```
                    <xsd:element minOccurs="0" name="Issuer" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="NotBefore" type="xsd:dateTime"/>
                    <xsd:element minOccurs="0" name="NotOnOrAfter" type="xsd:dateTime"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAssertionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="SAMLAssertion" type="tns:SAMLAssertion"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAttributeAssertion">
        <xsd:complexContent>
            <xsd:extension base="tns:SAMLAssertion">
                <xsd:sequence>
                    <xsd:element minOccurs="0"
                        name="AttributeStatements"
type="tns:SAMLAttributeStatementList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAttributeAssertionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                      name="SAMLAttributeAssertion" type="tns:SAMLAttributeAssertion"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAttributeStatement">
        <xsd:complexContent>
            <xsd:extension base="tns:SAMLSubjectStatement"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAttributeStatementList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                      name="SAMLAttributeStatement" type="tns:SAMLAttributeStatement"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAuthenticationAssertion">
        <xsd:complexContent>
            <xsd:extension base="tns:SAMLAssertion">
```

```
                <xsd:sequence>
                    <xsd:element minOccurs="0"
                        name="AuthenticationStatements"
type="tns:SAMLAuthenticationStatementList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAuthenticationAssertionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="SAMLAuthenticationAssertion"
type="tns:SAMLAuthenticationAssertion"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAuthenticationStatement">
        <xsd:complexContent>
            <xsd:extension base="tns:SAMLSubjectStatement">
                <xsd:sequence>
                    <xsd:element minOccurs="0"
                        name="AuthenticationInstant" type="xsd:dateTime"/>
                    <xsd:element minOccurs="0"
                        name="AuthenticationMethod" type="xsd:anyURI"/>
                    <xsd:element minOccurs="0" name="DNSAddress" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="IPAddress" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAuthenticationStatementList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="SAMLAuthenticationStatement"
type="tns:SAMLAuthenticationStatement"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAuthorizationDecisionAssertion">
        <xsd:complexContent>
            <xsd:extension base="tns:SAMLAssertion">
                <xsd:sequence>
                    <xsd:element minOccurs="0"
                        name="AuthorizationDecisionStatements"
type="tns:SAMLAuthorizationDecisionStatementList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAuthorizationDecisionAssertionList">
```

```
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="SAMLAuthorizationDecisionAssertion"
type="tns:SAMLAuthorizationDecisionAssertion"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAuthorizationDecisionStatement">
        <xsd:complexContent>
            <xsd:extension base="tns:SAMLSubjectStatement"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLAuthorizationDecisionStatementList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="SAMLAuthorizationDecisionStatement"
type="tns:SAMLAuthorizationDecisionStatement"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLSubjectAssertion">
        <xsd:complexContent>
            <xsd:extension base="tns:SAMLAssertion">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="SubjectStatements"
type="tns:SAMLSubjectStatementList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLSubjectAssertionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="SAMLSubjectAssertion" type="tns:SAMLSubjectAssertion"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SAMLSubjectStatement">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="SecurityDomain" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="SubjectName" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
```

```
    <xsd:complexType mixed="true" name="SAMLSubjectStatementList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="SAMLSubjectStatement" type="tns:SAMLSubjectStatement"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Signature">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SignatureList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Signature" type="tns:Signature"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="TrustedIdentity">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="CACerts" type="tns:CACertList"/>
                    <xsd:element minOccurs="0" name="CertType"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="DataType"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="DegreeOfSeparation"
type="xsd:long"/>
                    <xsd:element minOccurs="0" name="IdentityAddress"
type="core:IdentityAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="TrustedIdentityList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="TrustedIdentity" type="tns:TrustedIdentity"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Service: Data

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="Data" targetNamespace="xns:+xns/Data"
    xmlns:core="xns:+xns/Core" xmlns:tns="xns:+xns/Data"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Core"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
    <xsd:complexType mixed="true" name="Add">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Before" type="core:DataID"/>
                    <xsd:element minOccurs="0" name="ContractID" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="DataPath" type="core:DataID"/>
                    <xsd:element minOccurs="0" name="Object" type="core:XNSObject"/>
                    <xsd:element minOccurs="0" name="Parent" type="core:DataID"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AddRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Add"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AddResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Add"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ChangeNotify">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Changes" type="core:DataIDList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ChangeNotifyRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:ChangeNotify"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ChangeNotifyResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:ChangeNotify"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Delete">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
```

```
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="ContractID" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="DataAddress"
type="core:XNSAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DeleteRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Delete"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DeleteResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Delete"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Get">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Addresses" type="core:TextList"/>
                    <xsd:element minOccurs="0" name="Data" type="core:ElementList"/>
                    <xsd:element minOccurs="0" name="Depth" type="xsd:long"/>
                    <xsd:element minOccurs="0" name="IncludeTypes" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Limit" type="xsd:long"/>
                    <xsd:element minOccurs="0" name="Skip" type="xsd:long"/>
                    <xsd:element minOccurs="0" name="Type" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Get"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Get"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetListElements">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Address" type="core:XNSAddress"/>
                    <xsd:element minOccurs="0" name="Data" type="core:ElementList"/>
                    <xsd:element minOccurs="0" name="Depth" type="xsd:long"/>
                    <xsd:element minOccurs="0" name="IncludeTypes" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Limit" type="xsd:long"/>
                    <xsd:element minOccurs="0" name="Skip" type="xsd:long"/>
                    <xsd:element minOccurs="0" name="Type" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
```

```
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="GetListElementsRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:GetListElements"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="GetListElementsResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:GetListElements"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="GetVersions">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="Address" type="core:XNSAddress"/>
                <xsd:element minOccurs="0" name="Versions" type="core:RefList"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="GetVersionsRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:GetVersions"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="GetVersionsResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:GetVersions"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="Increment">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="Address" type="core:XNSAddress"/>
                <xsd:element minOccurs="0" name="Amount" type="xsd:long"/>
                <xsd:element minOccurs="0" name="ElementName" type="xsd:string"/>
                <xsd:element minOccurs="0" name="IncrementedValue" type="xsd:long"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="IncrementRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:Increment"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="IncrementResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:Increment"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RespondToUpdate">
    <xsd:complexContent>
```

```
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Address"
type="core:IdentityDataID"/>
                    <xsd:element minOccurs="0" name="Args" type="core:TextList"/>
                    <xsd:element minOccurs="0" name="DataType"
type="core:IdentityDataAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RespondToUpdateRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:RespondToUpdate"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RespondToUpdateResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:RespondToUpdate"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Set">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Changes"
type="tns:ChangeRequestList"/>
                    <xsd:element minOccurs="0" name="ContractID" type="xsd:string"/>
                   <xsd:element minOccurs="0" name="NewDataPaths" type="core:TextList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SetRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Set"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SetResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Set"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Update">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="AttrName" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="ContractID" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="DataPath" type="core:DataID"/>
                    <xsd:element minOccurs="0" name="ElemName" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Value" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
```

```
    <xsd:complexType mixed="true" name="UpdateRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Update"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="UpdateResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Update"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="UpdateObject">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Address"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="Object" type="core:XNSObject"/>
                    <xsd:element minOccurs="0" name="UpdateObject" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="UpdateObjectRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:UpdateObject"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="UpdateObjectResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:UpdateObject"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ChangeRequest">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Address" type="core:DataID"/>
                    <xsd:element minOccurs="0" name="Attr" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Before" type="core:DataID"/>
                    <xsd:element minOccurs="0" name="Object" type="core:XNSObject"/>
                    <xsd:element minOccurs="0" name="Parent" type="core:DataID"/>
                    <xsd:element minOccurs="0" name="Tag" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Type" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Value" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ChangeRequestList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="ChangeRequest" type="tns:ChangeRequest"/>
                </xsd:sequence>
            </xsd:extension>
```

```
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Dir">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Data" type="core:RefList"/>
                    <xsd:element minOccurs="0" name="Dirs"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DirList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Dir" type="tns:Dir"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Identity">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Data"
type="tns:TypeCollectionList"/>
                    <xsd:element minOccurs="0" name="Dir" type="tns:Dir"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="IdentityList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Identity" type="tns:Identity"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="TypeCollection">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Derivatives"
type="core:IdentityDataAddressList"/>
                    <xsd:element minOccurs="0" name="DerivedFrom"
type="core:IdentityDataAddress"/>
                   <xsd:element minOccurs="0" name="Instances" type="core:ElementList"/>
                    <xsd:element minOccurs="0" name="SchemaAddress"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="TypeName" type="xsd:string"/>
                   <xsd:element minOccurs="0" name="Versions" type="core:ElementList"/>
```

```
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="TypeCollectionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="TypeCollection" type="tns:TypeCollection"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Service: Discovery

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="Discovery" targetNamespace="xns:+xns/Discovery"
    xmlns:core="xns:+xns/Core" xmlns:tns="xns:+xns/Discovery"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Core"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
    <xsd:complexType mixed="true" name="DescribeService">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="HTMLLink" type="xsd:anyURI"/>
                   <xsd:element minOccurs="0" name="ServiceDef" type="tns:ServiceDef"/>
                    <xsd:element minOccurs="0" name="ServiceName" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="WSDLLink" type="xsd:anyURI"/>
                    <xsd:element minOccurs="0" name="XSDLink" type="xsd:anyURI"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DescribeServiceRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:DescribeService"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DescribeServiceResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:DescribeService"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetServiceProviders">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Providers"
type="core:IdentityAddressList"/>
                    <xsd:element minOccurs="0" name="Service"
type="core:IdentityDataAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetServiceProvidersRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:GetServiceProviders"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetServiceProvidersResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:GetServiceProviders"/>
        </xsd:complexContent>
    </xsd:complexType>
```

```
    <xsd:complexType mixed="true" name="GetServicesDefined">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Services" type="core:TextList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetServicesDefinedRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:GetServicesDefined"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetServicesDefinedResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:GetServicesDefined"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetServicesOffered">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="AuthorizedServices"
type="core:IdentityDataAddressList"/>
                    <xsd:element minOccurs="0" name="NegotiableServices"
type="core:IdentityDataAddressList"/>
                    <xsd:element minOccurs="0" name="PublicServices"
type="core:IdentityDataAddressList"/>
                    <xsd:element minOccurs="0" name="Terms" type="core:TextList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetServicesOfferedRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:GetServicesOffered"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetServicesOfferedResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:GetServicesOffered"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RegisterServiceProvider">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="ServiceProvider"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="Services"
type="core:IdentityDataAddressList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
```

```
    <xsd:complexType mixed="true" name="RegisterServiceProviderRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:RegisterServiceProvider"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RegisterServiceProviderResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:RegisterServiceProvider"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RetireServiceProvider">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="ServiceProvider"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="Services"
type="core:IdentityDataAddressList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RetireServiceProviderRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:RetireServiceProvider"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RetireServiceProviderResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:RetireServiceProvider"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataElemDef">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="DataType" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="PromoteNames" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Recurring" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Required" type="xsd:boolean"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataElemDefList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="DataElemDef" type="tns:DataElemDef"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataTypeDef">
        <xsd:complexContent>
```

```
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="DerivedFrom"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="ElemDefs"
type="tns:DataElemDefList"/>
                    <xsd:element minOccurs="0" name="EnumDefs" type="tns:EnumDefList"/>
                    <xsd:element minOccurs="0"
                        name="TypeCollectionAddress" type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="Version" type="xsd:long"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataTypeDefList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="DataTypeDef" type="tns:DataTypeDef"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Dependency">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Identity"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="Required" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="ServiceName" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DependencyList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Dependency" type="tns:Dependency"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="EnumDef">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="IsDefault" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="SequenceNo" type="xsd:long"/>
                    <xsd:element minOccurs="0" name="Value" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
```

```
    </xsd:complexType>
    <xsd:complexType mixed="true" name="EnumDefList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="EnumDef" type="tns:EnumDef"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MessageDef">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Args" type="tns:DataElemDefList"/>
                    <xsd:element minOccurs="0" name="DerivedFrom"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="Exceptions"
type="core:XNSObjectList"/>
                   <xsd:element minOccurs="0" name="PostConditions" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="PreConditions" type="xsd:string"/>
                  <xsd:element minOccurs="0" name="Return" type="tns:DataElemDefList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MessageDefList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="MessageDef" type="tns:MessageDef"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ServiceDef">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Data" type="tns:DataTypeDefList"/>
                    <xsd:element minOccurs="0" name="Dependencies"
type="tns:DependencyList"/>
                    <xsd:element minOccurs="0" name="Messages"
type="tns:MessageDefList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ServiceDefList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="ServiceDef" type="tns:ServiceDef"/>
```

```
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Service: Folder

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="Folder" targetNamespace="xns:+xns/Folder"
    xmlns:core="xns:+xns/Core" xmlns:tns="xns:+xns/Folder"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Core"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
    <xsd:complexType mixed="true" name="AddItem">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Item" type="core:XNSAddress"/>
                    <xsd:element minOccurs="0" name="ItemName" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Path" type="core:XNSName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AddItemRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:AddItem"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AddItemResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:AddItem"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Create">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Name" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Path" type="core:XNSName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CreateRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Create"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CreateResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Create"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Delete">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
```

```
                <xsd:element minOccurs="0" name="Path" type="core:XNSName"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="DeleteRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:Delete"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="DeleteResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:Delete"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RemoveItem">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="ItemName" type="xsd:string"/>
                <xsd:element minOccurs="0" name="Path" type="core:XNSName"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RemoveItemRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:RemoveItem"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RemoveItemResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:RemoveItem"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="Rename">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="NewPath" type="core:XNSName"/>
                <xsd:element minOccurs="0" name="Path" type="core:XNSName"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RenameRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:Rename"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RenameResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:Rename"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RenameItem">
```

```
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="ItemName" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="NewName" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Path" type="core:XNSName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RenameItemRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:RenameItem"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RenameItemResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:RenameItem"/>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Service: Hosting

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="Hosting" targetNamespace="xns:+xns/Hosting"
    xmlns:core="xns:+xns/Core" xmlns:data="xns:+xns/Data"
    xmlns:tns="xns:+xns/Hosting" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Core"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
    <xsd:import namespace="xns:+xns/Data"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Data.xsd"/>
    <xsd:complexType mixed="true" name="DeleteIdentity">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="IdentityAddr"
type="core:IdentityAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DeleteIdentityRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:DeleteIdentity"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DeleteIdentityResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:DeleteIdentity"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="HostIdentity">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Identity" type="data:Identity"/>
                    <xsd:element minOccurs="0" name="IdentityAddr"
type="core:IdentityAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="HostIdentityRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:HostIdentity"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="HostIdentityResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:HostIdentity"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Host">
        <xsd:complexContent>
```

```
            <xsd:extension base="tns:GroupOwner">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="IDSP" type="tns:IDSP"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="HostList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Host" type="tns:Host"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="IDSP">
        <xsd:complexContent>
            <xsd:extension base="tns:Group">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Host" type="tns:Host"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="IDSPList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="IDSP" type="tns:IDSP"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Service: ID

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="ID" targetNamespace="xns:+xns/ID"
    xmlns:core="xns:+xns/Core" xmlns:tns="xns:+xns/ID"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Core"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
    <xsd:complexType mixed="true" name="MapID">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Context" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Map" type="tns:Map"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MapIDRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:MapID"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MapIDResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:MapID"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Register">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="HostID" type="core:HostID"/>
                   <xsd:element minOccurs="0" name="IdentityID" type="core:IdentityID"/>
                    <xsd:element minOccurs="0" name="IsHost" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Locations" type="core:TextList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RegisterRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Register"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RegisterResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Register"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Resolve">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
```

```
            <xsd:sequence>
                <xsd:element minOccurs="0" name="Cached" type="xsd:boolean"/>
                <xsd:element minOccurs="0" name="HostID" type="core:HostID"/>
               <xsd:element minOccurs="0" name="IdentityID" type="core:IdentityID"/>
                <xsd:element minOccurs="0" name="Locations" type="core:TextList"/>
                <xsd:element minOccurs="0" name="NoCache" type="xsd:boolean"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="ResolveRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:Resolve"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="ResolveResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:Resolve"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="Retire">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
               <xsd:element minOccurs="0" name="IdentityID" type="core:IdentityID"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RetireRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:Retire"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RetireResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:Retire"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="CachedID">
    <xsd:complexContent>
        <xsd:extension base="tns:ID"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="CachedIDList">
    <xsd:complexContent>
        <xsd:extension base="core:ElementList">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0"
                    name="CachedID" type="tns:CachedID"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="ID">
    <xsd:complexContent>
```

```
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="HostID" type="core:HostID"/>
                    <xsd:element minOccurs="0" name="Locations" type="core:TextList"/>
                    <xsd:element minOccurs="0" name="RegisteredID" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Retired" type="xsd:boolean"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="IDList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="ID" type="tns:ID"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="IDMap">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Map" type="tns:Map"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="IDMapList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="IDMap" type="tns:IDMap"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Map">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MapList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Map" type="tns:Map"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Service: Name

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="Name" targetNamespace="xns:+xns/Name"
    xmlns:core="xns:+xns/Core" xmlns:tns="xns:+xns/Name"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Core"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
    <xsd:complexType mixed="true" name="CloseNamespace">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Namespace" type="core:XNSName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CloseNamespaceRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:CloseNamespace"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CloseNamespaceResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:CloseNamespace"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Extend">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="ExpirationDate"
type="xsd:dateTime"/>
                    <xsd:element minOccurs="0" name="Name" type="core:XNSName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ExtendRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Extend"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ExtendResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Extend"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetValidNamespaces">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
```

```
                    <xsd:element minOccurs="0" name="Namespaces"
type="core:XNSNameList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetValidNamespacesRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:GetValidNamespaces"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetValidNamespacesResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:GetValidNamespaces"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="OpenNamespace">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Namespace" type="core:XNSName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="OpenNamespaceRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:OpenNamespace"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="OpenNamespaceResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:OpenNamespace"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Register">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="ExpirationDate"
type="xsd:dateTime"/>
                    <xsd:element minOccurs="0" name="FullName" type="core:XNSName"/>
                    <xsd:element minOccurs="0" name="IdentityAddress"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="Name" type="core:XNSName"/>
                    <xsd:element minOccurs="0" name="ReservationID" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RegisterRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Register"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RegisterResponse">
```

```
    <xsd:complexContent>
        <xsd:extension base="tns:Register"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="Release">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="Name" type="core:XNSName"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="ReleaseRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:Release"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="ReleaseResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:Release"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="Rename">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="NewName" type="core:XNSName"/>
                <xsd:element minOccurs="0" name="OldName" type="core:XNSName"/>
                <xsd:element minOccurs="0" name="ReservationID" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RenameRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:Rename"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RenameResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:Rename"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="Request">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="Name" type="core:XNSName"/>
                <xsd:element minOccurs="0" name="ReservationID" type="xsd:string"/>
                <xsd:element minOccurs="0" name="Reserve" type="xsd:boolean"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RequestRequest">
```

```
        <xsd:complexContent>
            <xsd:extension base="tns:Request"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="RequestResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Request"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Resolve">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Cached" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="ExpirationDate"
type="xsd:dateTime"/>
                    <xsd:element minOccurs="0" name="IdentityAddress"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="Name" type="core:XNSName"/>
                    <xsd:element minOccurs="0" name="NoCache" type="xsd:boolean"/>
                   <xsd:element minOccurs="0" name="PartialResolve" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="PartiallyResolved"
type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="RelativeRemainder"
type="core:XNSName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ResolveRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Resolve"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ResolveResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Resolve"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ResolveNamespace">
        <xsd:complexContent>
            <xsd:extension base="tns:Resolve">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Namespace" type="core:XNSName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ResolveNamespaceRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:ResolveNamespace"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ResolveNamespaceResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:ResolveNamespace"/>
```

```
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Transfer">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="IdentityAddress"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="Name" type="core:XNSName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="TransferRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Transfer"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="TransferResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Transfer"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CachedName">
        <xsd:complexContent>
            <xsd:extension base="tns:Name"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="CachedNameList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="CachedName" type="tns:CachedName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MyName">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="ExpirationDate"
type="xsd:dateTime"/>
                    <xsd:element minOccurs="0" name="RegisteredName"
type="core:XNSName"/>
                    <xsd:element minOccurs="0" name="Registrar"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="ReservationID" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MyNameList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
```

```
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="MyName" type="tns:MyName"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Name">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Address"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="ExpirationDate"
type="xsd:dateTime"/>
                    <xsd:element minOccurs="0" name="RegisteredName"
type="core:XNSName"/>
                    <xsd:element minOccurs="0" name="ReservationID" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="NameList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Name" type="tns:Name"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ServiceProfile">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="PreferredName" type="tns:MyName"/>
                    <xsd:element minOccurs="0" name="ValidNamespaces"
type="core:XNSNameList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ServiceProfileList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="ServiceProfile" type="tns:ServiceProfile"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Service: Negotiation

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="Negotiation" targetNamespace="xns:+xns/Negotiation"
    xmlns:certification="xns:+xns/Certification"
    xmlns:core="xns:+xns/Core" xmlns:tns="xns:+xns/Negotiation"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Core"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
    <xsd:import namespace="xns:+xns/Certification"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Certification.xsd"/>
    <xsd:complexType mixed="true" name="ConfirmNegotiation">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Accepted" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Contract" type="tns:Contract"/>
                    <xsd:element minOccurs="0" name="ContractName" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmNegotiationRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:ConfirmNegotiation"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmNegotiationResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:ConfirmNegotiation"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmReceiptAccepted">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Accepted" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="ReasonForRejection"
type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Reference" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmReceiptAcceptedRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:ConfirmReceiptAccepted"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ConfirmReceiptAcceptedResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:ConfirmReceiptAccepted"/>
        </xsd:complexContent>
```

```
</xsd:complexType>
<xsd:complexType mixed="true" name="NegotiateContract">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="Contract" type="tns:Contract"/>
                <xsd:element minOccurs="0" name="Negotiated" type="xsd:boolean"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="NegotiateContractRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:NegotiateContract"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="NegotiateContractResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:NegotiateContract"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RequestForm">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="Form" type="tns:Form"/>
                <xsd:element minOccurs="0" name="FormAddr" type="core:XNSAddress"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RequestFormRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:RequestForm"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="RequestFormResponse">
    <xsd:complexContent>
        <xsd:extension base="tns:RequestForm"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="SubmitForConfirmation">
    <xsd:complexContent>
        <xsd:extension base="core:Message">
            <xsd:sequence>
                <xsd:element minOccurs="0" name="Accepted" type="xsd:boolean"/>
                <xsd:element minOccurs="0" name="Contract" type="tns:Contract"/>
                <xsd:element minOccurs="0" name="ContractName" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType mixed="true" name="SubmitForConfirmationRequest">
    <xsd:complexContent>
        <xsd:extension base="tns:SubmitForConfirmation"/>
    </xsd:complexContent>
```

```
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitForConfirmationResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:SubmitForConfirmation"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitForNegotiation">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Contract" type="tns:Contract"/>
                    <xsd:element minOccurs="0" name="Form" type="tns:Form"/>
                    <xsd:element minOccurs="0" name="NegotiateWith"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="Negotiated" type="xsd:boolean"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitForNegotiationRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:SubmitForNegotiation"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitForNegotiationResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:SubmitForNegotiation"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitReceipt">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Accepted" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="OwnerRequired" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Receipt" type="tns:Receipt"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitReceiptRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:SubmitReceipt"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitReceiptResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:SubmitReceipt"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="TerminateContract">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="ContractName" type="xsd:string"/>
                </xsd:sequence>
```

```
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="TerminateContractRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:TerminateContract"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="TerminateContractResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:TerminateContract"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Contract">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="BeginDate" type="xsd:dateTime"/>
                    <xsd:element minOccurs="0" name="ContractDate" type="xsd:dateTime"/>
                    <xsd:element minOccurs="0" name="ContractID" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="ContractTerms"
type="tns:ContractTerms"/>
                    <xsd:element minOccurs="0" name="ContractType"
type="tns:ContractTypeEnum"/>
                    <xsd:element minOccurs="0"
                        name="DataAccessPermissions"
type="tns:DataAccessPermissionList"/>
                    <xsd:element minOccurs="0" name="DataSet"
type="tns:ContractDataSetList"/>
                    <xsd:element minOccurs="0" name="EndDate" type="xsd:dateTime"/>
                    <xsd:element minOccurs="0" name="Form" type="core:XNSAddress"/>
                    <xsd:element minOccurs="0" name="GroupID" type="core:DataID"/>
                    <xsd:element minOccurs="0" name="MessagePermissions"/>
                    <xsd:element minOccurs="0" name="Originator"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="Pending" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="PrivacyPermissions"/>
                    <xsd:element minOccurs="0" name="Provider"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="SyncPermissions"/>
                    <xsd:element minOccurs="0" name="TerminatedBy"
type="core:IdentityID"/>
                    <xsd:element minOccurs="0" name="TerminationDate"
type="xsd:dateTime"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ContractList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Contract" type="tns:Contract"/>
                </xsd:sequence>
            </xsd:extension>
```

```
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ContractDataSet">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Data" type="core:XNSObjectList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ContractDataSetList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="ContractDataSet" type="tns:ContractDataSet"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ContractTerms">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="HTMLPrivacyPolicy"
type="xsd:string"/>
                    <xsd:element minOccurs="0" name="HTMLSecurityPolicy"
type="xsd:string"/>
                  <xsd:element minOccurs="0" name="HTMLTermsPolicy" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="P3PPrivacyPolicy"
type="xsd:string"/>
                    <xsd:element minOccurs="0" name="PurposeText" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ContractTermsList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="ContractTerms" type="tns:ContractTerms"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="ContractTypeEnum">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="EXTENSION"/>
            <xsd:enumeration value="SUBSCRIPTION"/>
            <xsd:enumeration value="SERVICE"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="DataAccessPermission">
        <xsd:complexContent>
```

```
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="CanCreate" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="CanDelete" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="CanRead" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="CanUpdate" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="DataIDs" type="core:DataIDList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataAccessPermissionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="DataAccessPermission" type="tns:DataAccessPermission"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataDistribution">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Data" type="core:ElementList"/>
                    <xsd:element minOccurs="0" name="DistributeTo"
type="core:IdentityDataAddressList"/>
                    <xsd:element minOccurs="0" name="SchemaAddress"
type="core:IdentityDataAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataDistributionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="DataDistribution" type="tns:DataDistribution"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataPermission">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="DataNames" type="core:TextList"/>
                    <xsd:element minOccurs="0" name="Purpose" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataPermissionList">
        <xsd:complexContent>
```

```
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="DataPermission" type="tns:DataPermission"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataPermissionRequest">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                   <xsd:element minOccurs="0" name="DataRequests" type="core:TextList"/>
                    <xsd:element minOccurs="0" name="OptOut" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Purpose" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Required" type="xsd:boolean"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataPermissionRequestList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="DataPermissionRequest" type="tns:DataPermissionRequest"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataRequest">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="DataType"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="RequestorName" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="Required" type="xsd:boolean"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="DataRequestList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="DataRequest" type="tns:DataRequest"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Form">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
```

```
                    <xsd:element minOccurs="0" name="ContractTerms"
type="tns:ContractTerms"/>
                    <xsd:element minOccurs="0" name="ContractType"
type="tns:ContractTypeEnum"/>
                    <xsd:element minOccurs="0"
                        name="DataAccessPermissions"
type="tns:DataAccessPermissionList"/>
                    <xsd:element minOccurs="0" name="DataRequests"
type="tns:DataRequestList"/>
                    <xsd:element minOccurs="0" name="IdentityNegotiate"
type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="MessagePermissions"/>
                    <xsd:element minOccurs="0" name="PrivacyRequests"/>
                    <xsd:element minOccurs="0" name="SyncRequests"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="FormList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Form" type="tns:Form"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Link">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Contracts"/>
                    <xsd:element minOccurs="0" name="Identity"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="KnownAs"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="PublicKey"
type="certification:PublicKeyCert"/>
                    <xsd:element minOccurs="0" name="Receipts" type="tns:ReceiptList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LinkList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Link" type="tns:Link"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MessagePermission">
        <xsd:complexContent>
```

```
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Condition" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="MsgName"
type="core:IdentityDataAddress"/>
                    <xsd:element minOccurs="0" name="ProcessStates"
type="core:TextList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="MessagePermissionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="MessagePermission" type="tns:MessagePermission"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="PermissionTypeEnum">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="CONTACT"/>
            <xsd:enumeration value="DISCLOSURE"/>
            <xsd:enumeration value="RETENTION"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="PrivacyPermission">
        <xsd:complexContent>
            <xsd:extension base="tns:DataPermission">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Agreed" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Implied" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Party" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="PermissionType"
type="tns:PermissionTypeEnum"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="PrivacyPermissionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="PrivacyPermission" type="tns:PrivacyPermission"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="PrivacyPermissionRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:DataPermissionRequest">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Party" type="xsd:string"/>
```

```
                    <xsd:element minOccurs="0" name="PermissionType"
type="tns:PermissionTypeEnum"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="PrivacyPermissionRequestList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="PrivacyPermissionRequest"
type="tns:PrivacyPermissionRequest"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Receipt">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="Contract" type="xsd:string"/>
                   <xsd:element minOccurs="0" name="Details" type="tns:ReceiptDetail"/>
                    <xsd:element minOccurs="0" name="Pending" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="Reference" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="TrxDate" type="xsd:dateTime"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ReceiptList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="Receipt" type="tns:Receipt"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ReceiptDetail">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ReceiptDetailList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="ReceiptDetail" type="tns:ReceiptDetail"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="SecurityLevelEnum">
```

```
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="NONE"/>
            <xsd:enumeration value="HTTPS_40"/>
            <xsd:enumeration value="HTTPS_128"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="SyncLevelEnum">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="ONE_TIME"/>
            <xsd:enumeration value="PUSH"/>
            <xsd:enumeration value="PUSH_OR_PULL"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType mixed="true" name="SyncPermission">
        <xsd:complexContent>
            <xsd:extension base="tns:DataPermission">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="PushOnChange" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="SecurityLevel"
type="tns:SecurityLevelEnum"/>
                    <xsd:element minOccurs="0" name="SyncLevel"
type="tns:SyncLevelEnum"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SyncPermissionList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="SyncPermission" type="tns:SyncPermission"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SyncPermissionRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:DataPermissionRequest">
                <xsd:sequence>
                    <xsd:element minOccurs="0"
                        name="PushOnChangeRequired" type="xsd:boolean"/>
                    <xsd:element minOccurs="0" name="SecurityProposals"
type="core:TextList"/>
                    <xsd:element minOccurs="0" name="SyncProposals"
type="core:TextList"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SyncPermissionRequestList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="SyncPermissionRequest" type="tns:SyncPermissionRequest"/>
```

```
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Service: Session

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="Session" targetNamespace="xns:+xns/Session"
    xmlns:core="xns:+xns/Core" xmlns:tns="xns:+xns/Session"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
    <xsd:import namespace="xns:+xns/Core"
schemaLocation="http://specs.onename.com/specs/xns/gen/xns/Core.xsd"/>
    <xsd:complexType mixed="true" name="Authenticate">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="LogoutURI" type="xsd:anyURI"/>
                    <xsd:element minOccurs="0" name="MerchantAddr"
type="core:IdentityAddress"/>
                    <xsd:element minOccurs="0" name="MerchantName" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="MerchantURI" type="xsd:anyURI"/>
                    <xsd:element minOccurs="0" name="PreferredAA" type="xsd:anyURI"/>
                    <xsd:element minOccurs="0" name="SessionURI" type="xsd:anyURI"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AuthenticateRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Authenticate"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="AuthenticateResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Authenticate"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetSessionURI">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="SessionURI" type="xsd:anyURI"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetSessionURIRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:GetSessionURI"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="GetSessionURIResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:GetSessionURI"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="Login">
```

```
        <xsd:complexContent>
            <xsd:extension base="tns:Authenticate"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LoginRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:Login"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LoginResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:Login"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LoginNotify">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="AuthCert" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LoginNotifyRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:LoginNotify"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LoginNotifyResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:LoginNotify"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LogoutNotify">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="RedirectURI" type="xsd:anyURI"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LogoutNotifyRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:LogoutNotify"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="LogoutNotifyResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:LogoutNotify"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitAuthCert">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
```

```
                    <xsd:element minOccurs="0" name="AuthCert" type="xsd:string"/>
                    <xsd:element minOccurs="0" name="AuthIdentity"
type="core:IdentityAddress"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitAuthCertRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:SubmitAuthCert"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="SubmitAuthCertResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:SubmitAuthCert"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="XNSSessionLogout">
        <xsd:complexContent>
            <xsd:extension base="core:Message">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="RedirectURI" type="xsd:anyURI"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="XNSSessionLogoutRequest">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSSessionLogout"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="XNSSessionLogoutResponse">
        <xsd:complexContent>
            <xsd:extension base="tns:XNSSessionLogout"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ServiceProfile">
        <xsd:complexContent>
            <xsd:extension base="core:XNSObject">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="SessionURI" type="xsd:anyURI"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType mixed="true" name="ServiceProfileList">
        <xsd:complexContent>
            <xsd:extension base="core:ElementList">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="ServiceProfile" type="tns:ServiceProfile"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

APPENDIX C

Glossary (Normative)

The subject matter of digital identity demands precise technical terminology not only because it deals with the highest levels of abstraction, but also because the real-world terminology that is most relevant to this field is already among the most general and abstract—and therefore fuzzy and easily confused—of any of the words in our language.

For these reasons the contributors to the XNS Technical Specifications have developed the following glossary and made it normative to the specifications. Feedback and suggestions are welcome via the forums at the XNSORG web site (www.xns.org).

Absolute address In network addressing, an addressing value which is globally unique in the addressing system and which does not require any predefined context to be resolved. In XNS, the globally unique address of any XNS-addressable node within an identity document. An XNS absolute address consists of an identity address plus an optional data address. Note that while every absolute address is unique, the ability of an XNS identity to register any number of identity IDs or names means an XNS-addressable node may have more than one absolute address. An XNS absolute address may contain either an absolute ID or an absolute name.

Absolute ID The persistent globally unique identifier of any XNS-addressable node within an identity document. An absolute ID consists of an identity ID plus an optional data ID. Once assigned, an absolute ID will never change–if the node it addresses is deleted, the ID will be retired. Note that while each absolute ID is unique, the ability of an XNS identity to register any number of identity IDs means an XNS-addressable node may have more than one absolute ID. See also absolute address.

Absolute name The globally unique name of any XNS-addressable node within an identity document. An absolute name consists of an identity name plus an optional data name. It is not required for any XNS-addressable node to have an absolute name, and absolute names may change during the lifetime of a node. Absolute names are resolved to absolute IDs. See also absolute address and absolute ID.

Address A value that can be uniquely associated with a node on a communications network. An address is either a locator or a location.

Anonymity The ability for a digital identity to be asserted without any association to the real world identity of the principal it represents nor any association with other digital identities or identifiers that may represent the principal.

Attribute In object-oriented design, a datatype associated with an object. In identity management, a datatype instance that describes an identity. In XNS, a general term for an instance of an XNS datatype definition stored in an identity document to describe the properties of the identity. See identity attribute.

Attribute authority A service provider who offers XNS Data service for any set of attributes stored in an identity document.

Attribute group A collection of attributes. In XNS, a XNS datatype that includes other XNS datatypes. An attribute group is often called a profile.

Authentication The process of an identity controller asserting one or more credentials to prove that they are the rightful controller of an identity.

Authorization The process of an identity controller asserting one or more credentials to prove that an identity is authorized to access a network resource (including another identity).

Branch In network addressing, the collection of all nodes in all paths that begin from a starting node in a tree structure. The starting node is called the root of the branch.

Business identity See Organizational identity.

Certification The process of obtaining a digital certificate from a certification authority.

Certification authority A service provider who offers the service of providing digital certificates.

Community identity The XNS identity responsible for establishing the hosting services, schema and service definitions, and other common relationships shared by a group of XNS identities. A common type of community identity is a host identity.

Community service An XNS service offered by a community identity.

Connector In network architecture (especially systems integration), a system component used to form a communications channel between two systems that speak the same communications protocol. In XNS, a system component used to form a communications channel between two XNS identities. Specified by a URI.

Contract See XNS contract.

Credential An attribute or attribute group of an identity that make an assertion about other attributes of that identity, such as the validity of a public key or a driver's license. A common form of credential is a digital certificate.

Cross-reference (X-ref) In XNS, a reference from one identity or identity attribute (i.e., an XNS object) to another identity or identity attribute (i.e., another XNS object) that represents the same principal or principal attribute. Always expressed as an absolute address. An XNS object may have zero or more cross-references. Cross-references are stored in the global XNSXRefs element of an XNS object.

Current context In network addressing, the starting node for resolving a relative addressing value.

Data address An XNS address that can be resolved entirely within a single identity document. May consist of either a Data ID or a Data name.

Data ID A unique persistent ID value for any XNS-addressable node within an identity document that will never change for the lifetime of that node. It consists of the path of XNS ID attribute values from the root node of the identity document to the target node. Every XNS-addressable node within an identity document has at least one data ID, however through the use of references a node can have more than one data ID.

Data name A unique non-persistent name value for any XNS-addressable node within an identity document. It consists of the path of XNS name attribute values from the root node of the document to the target node. An XNS-addressable node may have zero or more data names; all data names are controlled by the identity controller and may change over time.

Data schema A particular kind of schema representing the organization of data elements.

Digital certificate An electronic credential that uses digital signatures to make an assertion about the attributes of an identity. X.509 is one of the most common digital certificate formats.

Digital identity A representation of a real-world identity in electronic form. See Identity and Web identity.

Digital signature A collection of bits attached to an electronic document or message that cryptographically validates the digital identity of the signer.

Directory A hierarchical system for identifying and organizing resources and their attributes.

Directory entry A node in a directory tree representing an identity. Contrast with Web identity.

Directory tree The hierarchical organization of a directory, e.g., the X.500 directory system. Contrast with identity tree and identity web.

Document root In XML, the root node of an XML document. In XNS, the root node of an identity document.

Document web A set of linked hypertext pages. Contrast with identity web. The HTML, HTTP, and URL specifications permitted the creation of the world's largest document web, the World Wide Web.

Domain In network architecture, the collection of resources under common administration, including a shared security environment. In XNS, the collection of XNS identities that share a common legal owner. A specialization of group. The three types of XNS domains match the three types of XNS identities: personal domains represent individuals, organizational domains represent any legal entity other than an individual, and general domains represents public taxonomies managed by XNSORG or its delegates.

Domain member In XNS, any identity that belongs to a domain, and thus is owned by the domain owner. A specialization of group member.

Domain owner In XNS, the legal owner of a domain. A specialization of group owner.

Extended identity Any of a set of identities that represents the same principal in different domains. The identity definition of an extended identity contains the identity address of each identity it is extending as a cross-reference.

Federation In network addressing, the ability to create addresses that can span multiple administrative domains.

General identity In XNS, a type of identity that represents any generic category, entity, or object whose identity is defined by linguistic, cultural, or scientific convention.

Global community See XNS global community.

Global community service A community service offered by XNSORG on behalf of the XNS community to facilitate the global interoperability of XNS.

Group In XNS, a collection of identities that have something in common. A group is to identities what a directory is to data. Every group has a group owner and zero or more group members. Groups can be hierarchical, i.e., a group can be a member of another group. Two specializations of group used in XNS architecture are host community and domain.

Group definition The XNS object that defines a group, including its attributes and members. A group definition is typically contained by the identity representing the group owner, but it may also be stored in another identity.

Group member An identity that belongs to a group.

Group owner The identity that owns the definition of a group by controlling the group definition.

Host community The collection of XNS identities that share a common network endpoint. The identity of the endpoint is represented by a host identity. All the other identities at this endpoint are hosted identities. A specialization of group.

Host identity Any XNS identity that represents the identity of a network endpoint. Host identities provide XNS Hosting service to other identities at the same network endpoint. A specialization of group owner.

Hosted identity Any identity receiving XNS Hosting service from a host identity. All identities in a host community besides the host identity are hosted identities.

Hosting service An XNS base service offered by a host identity to the hosted identities in the same host community. XNS Hosting services include creating and deleting identity documents.

ID Identifier. In general, a unique value associated with an identity. In XNS, an immutable persistent non-semantic addressing value stored in the global XNS ID attribute of every XNS object. Every ID value is unique at least in the ID space of its parent object.

ID path In XNS addressing, the path formed by the values of the global XNS ID attribute of each XNS object representing a node in the address.

ID space In XNS addressing, the set of all XNS ID attribute values of XNS objects that are children of the parent XNS object defining the ID space.

Identity In XNS, an independent encapsulation of attribute values that is uniquely addressable. "Independent" means capable of participating in non-exclusive relationships with other identities. In formal UML, this means objects that are not inherently limited to composition (whole/part) relationships, but which can participate in aggregation (ownership) relationships. In XNS, identities are one of three types: personal identities, organizational identities, or general identities, depending on their identity controller.

Identity address An XNS address that terminates at the root node of an identity document. An identity address is always an absolute address.

Identity agent In XNS, the software process that acts upon an identity document to provide identity services to an identity controller.

Identity attribute In identity management, a datatype instance that describes an identity. In XNS, an XNS object contained in an identity document.

Identity controller The real-world entity responsible for the management of an XNS identity. In XNS, identity controllers are one of three types: an individual, a legal organization, or the general public as represented by XNSORG or its delegate. An identity controller is only the same as the identity principal if the identity represents an individual and that individual is the identity controller. Personal identities can be controlled by individuals other than the principal (e.g., a parent may control a young child's identity), or an organization may control a personal identity used in a business context (e.g., an employee). For organizational identities, one or more personal identities may be the identity controller, and for general identities, one or more organizational identities may be the identity controller.

Identity credentials The credentials used to establish and assert the validity of a digital identity or its attributes.

Identity definition The singleton XNS object in every XNS identity document that reference the attributes or attribute groups that define the identity type of which this identity is an instance. Identity definitions are use to create and manage taxonomies.

Identity document The logical XML document that represents an identity in XNS. The schema for an identity document is defined by XNS Data service; other XNS schemas define data instances that can be stored in an identity document (attributes or attribute groups).

Identity federation The ability to create identities and identity addresses that can span multiple domains (spheres of legal ownership). A key architectural feature of XNS.

Identity ID A globally unique ID value associated with an XNS identity. Technically, it resolves to the root node of an identity document. An XNS identity ID is either a URN or an OID that stems from the XNS global community ID service. Note that while every identity ID is unique, for anonymity or pseudonymity purposes, an XNS identity may have any number of identity IDs.

Identity linking The process of forming a link between two identities that forms a protected data exchange relationship between them.

Identity name An XNS name that resolves to the root node of an identity document.

Identity server A software program used to host XNS identities and provide identity services to identity controllers. (The term may also be used to refer to the hardware on which such a program runs.) An identity server is not synonymous with an XNS host community; a single host community may span multiple identity servers, or multiple host communities may be operated on a single identity server. This is analogous to the difference between a web server and a web site.

Identity service Any of the set of services offered by an XNS identity on behalf of its identity controller to provide digital identity management.

Identity service provider In XNS, a service provider who offers identity services to a host community via a host identity.

Identity transaction An exchange of identity attributes from one identity to another under terms and conditions agreed to by both parties. See XNS link and XNS contract.

Identity transaction form A special XNS datatype used to define an identity transaction. Essentially a template for an XNS contract.

Identity tree A general term for the XML tree of structured attributes (datatype instances) represented in an identity document. Contrast with identity web, which is the "forest" created by linking individual identity trees, and directory tree, which is the purely hierarchical structure of a directory system.

Identity type A subelement of an identity definition used to establish the three types of identity controllers in XNS-personal, organizational, and general.

Identity web A collection of linked identity documents regardless of their location on the network. Contrast with document web. The XNS protocol can be used to create any number of identity webs; as these link together they will form a global Identity Web much like the World Wide Web.

Identity-relative address See data address. An XNS address that is relative to the root node of an identity document, i.e., it can be resolved entirely within that identity document.

IDSP See Identity service provider.

Independent address An XNS address in which the XNS ID portion is an independent path.

Independent object In XNS, an object that is contained directly in a type collection, i.e., no references must be traversed in the path from the identity document root to the target object. Contrast with referenced object.

Independent path In XNS addressing, the one and only hierarchical ID path from the root node of the address directly to the target node. An independent path will never change as long as the target node still exists, whereas a referenced path may be broken if the object containing the reference is modified.

Leaf In a tree structure, the final node in any path beyond which there are no further nodes.

Legal endpoint In XNS, any identity that belongs to a domain, i.e., is owned by the same legal owner. See domain.

Link In computer science, an association between two resources on a network that has associated behavior or operations. Contrast with reference. In XNS, an association between two XNS identities represented by an XNS link object. A link can contain one or more XNS contracts.

Link object In XNS, the datatype used to instantiate a link between two identities. A link object is a container for any number of XNS contract objects.

Location A node on a communications network identifiable by a unique addressing value in the native addressing scheme used by that network. For example, a postal address is a location on a postal network; a telephone number is a location on a telephone network; and an IP address is a location on the Internet. Note that a location on one communications network may be a locator for a lower-level communications network. For example, a URL is a location on the Web, but a locator for an IP address on the Internet, and in turn an IP address is a location on the Internet, but a locator for a machine-level network address (e.g., an Ethernet address). In XNS, a location is a list of URIs.

Locator An addressing value that must be resolved in order to determine a location. See also address and resolution.

Master identity In a set of extended identities (identities in different domains which all represent the same identity), the identity to which the identity controller chooses to initially direct XNS service requests, e.g., authentication and transaction messages.

Metaschema A schema that defines how to define other schemas. XNS employs a metaschema architecture defined through XNS Core and Discovery services. See also schema.

Name In general, a semantic addressing value. In XNS, a non-persistent semantic addressing value stored in the global XNS Name attribute of an XNS object. Every name value is unique at least in the namespace of its parent object.

Name path In XNS addressing, the path formed by the values of the global XNS Name attribute of each XNS object representing a node in the address.

Namespace In addressing, the set of all child nodes of a parent node. In XNS addressing, the set of all XNS Name attribute values of XNS objects that are children of the parent XNS object defining the namespace.

Network endpoint Any address on a data communications network to which a message can be sent. On the Internet, network endpoints are called hosts. In XNS, the identity of a network endpoint is represented by a host identity.

Node Generally, a point in a directed graph (e.g., a tree structure). In networking, an endpoint in a communications network. In addressing, a value in a path. In XML, an element or attribute in the tree structure of an XML document. In XNS, an XNS object. See also XNS-addressable node.

OID Object ID. One or more unique integers used in a dot-delimited path starting from a root node to form a unique ID for each object in the path. OIDs are commonly used in directory systems and standards like LDAP because produce smaller and more manageable addressing values than UUIDs.

Ontology A description or definition of the relationships between a community of objects or concepts. Similar to a taxonomy, but richer and not limited to hierarchical relationships.

Organizational identity In XNS, a type of identity that represents the identity of any legal entity besides a natural-born person. Organizational identities include sole proprietorships, partnerships, corporations, non-profits, governments, public host communities, academic institutions, etc. Also called a business identity.

Path A set of addressing values separated by one or more delimiters to form a larger addressing value. Each addressing value in a path is a node.

Personal identity In XNS, a type of identity that represents the identity of a natural-born person.

Principal The real-world entity represented by a digital identity, such as a person, an organization, or a generic object. A single principal may be represented by multiple digital identities on a network. A principal is only the same as an identity controller if the identity represents the same real-world person as controls the identity. For organizational identities, one or more personal identities may be the identity controller, and for general identities, one or more organizational identities may be the identity controller.

Profile In directory systems, a particular group of attributes associated with an identity. This term is used loosely in XNS because XNS allows for identities to have any number of attribute groups that would constitute a "profile".

Pseudonymity The ability for a digital identity to be asserted across a set of relationships defined by the identity controller without any association to the real world identity of the principal it represents.

Public data In XNS, XNS data that is not subject to access control and does not require an XNS link and XNS contract for another identity to access.

Public services In XNS, XNS services that are not subject to access control and do not require an XNS link and XNS contract for another identity to access.

Reference In computer science, a pointer from one object or resource to another. In XNS, a pointer from one XNS object to another. The referenced object or resource is called the target; the referencing object or resource is called the source. Unlike a link, a reference typically has no behavior or operations associated with it.

Referenced address An XNS address in which the XNS ID portion is a referenced path.

Referenced object In XNS, an object that is referenced by another object rather than contained in a parent/child relationship. Contrast with independent object.

Referenced path In XNS addressing, any path to a target object that includes a reference, i.e., indicates an aggregation relationship with the parent object. There may be any number of referenced paths to a target object. A referenced path may break if the object containing the reference is modified, whereas an independent path will never change as long as the target object exists.

Relative address In network addressing, an addressing value or path that can only be resolved relative to a given node in the addressing network, usually the current context.

Relative data address A data address that can only be resolved relative to the current context.

Reputation The process of obtaining a digital certificate asserting a value judgment regarding the attributes of an identity.

Reputation authority A service provider offers the service of providing reputation services.

Resolution The process of determining the location on a communications network associated with a locator. Only addresses that are locators need resolution; an address that is already a location does not need resolution because it is already in the native addressing scheme of its network.

Resolver The software process that performs resolution.

Root The node in a path above which there are no higher nodes.

Schema In computer science, a definition of the relationship between a set of objects or data elements. In XML, a definition of the valid structure of the elements, attributes, and other components of an XML document. In XNS, a definition of an XNS datatype, message type, or service type.

Scheme Closely related to schema, but often used for simpler forms of data relationships, such as the components of an addressing format.

Semantic abstraction The separation of non-persistent semantic identifiers (XNS names) from persistent abstract identifiers (XNS IDs). Semantic abstraction is what allows XNS to model long-term identity relationships (XNS links) that do not break even when the real world attributes of identities (names, telephone numbers, email addresses, etc.) change.

Service A set of messages and related datatypes that can be processed and returned by a process on a communications network in order to perform work for the service requestor.

Service consumer The software program necessary to access a service from a service provider. May also mean the principal using such software program.

Service implementation The software code necessary to provide a service.

Service provider The real-world legal entity (person, company, organization, government agency, etc.) responsible for providing a service.

Session An authentication or authorization relationship established between a service and a service consumer for a period of time. Typically represented by a session credential.

SOAP Simple Object Access Protocol. An XML messaging protocol for web services being standardized by the W3C.

Source In computer science, the object from which a reference or a link originates. The opposite of target.

Target In computer science, the object to which a reference or a link points. In addressing, the last node in a path. The opposite of source.

Target identity The identity to which an XNS address points.

Taxonomy A hierarchical classification system. The Yellow Pages, SIC codes, and the Dewey Decimal System are all taxonomies.

Type collection The special XNS object used to store instances of other XNS objects that are all of the same XNS datatype.

UDDI Universal Description, Discover, and Integration. A distributed directory service for web service description listings (see WSDL) being standardized by UDDI.org.

URI Uniform Resource Identifier. An IETF standard for a World Wide Web address that can be either a URL or a URN.

URL Uniform Resource Locator. A type of URI that serves as a current locator for resources on the World Wide Web.

URN Uniform Resource Name. A type of URI that provides a persistent address for a resource on the World Wide Web. A URN is typically resolved by a URN service to the current URL of the resource.

UUID Universal Unique ID. A 36 hex character format for globally unique identifiers that can be independently generated without the need for a central registration authority. The UUID format and algorithms were originally developed for DCE to provide object identifiers in a distributed computing environment.

Veronymity The ability for a digital identity to be associated with the real world identity of the principal it represents. This is usually accomplished by the identity controller providing credentials asserting the principal's legal identity.

Web identity An identity in a peer-to-peer web architecture where identities form links with other identities the same way Web pages link to other Web pages. A web identity is globally unique and does not dependent on any external hierarchy, but can participate in any number of hierarchies via identity linking. Contrast with directory entry.

Web service A service whose interface is described in XML in order to be widely interoperable. The most widely recognized Web services standards include SOAP, WSDL, and UDDI. XNS is an example of a Web service used for identity and relationship management.

WSDL Web Services Description Language. A XML format for publishing machine-processable interface descriptions for web services being standardized by the W3C.

XLink In XML, a standard for describing links between XML documents. In XNS, a global element of an XNS object that defines a synchronization relationship with another XNS object.

XNS Extensible Name Service. A protocol and infrastructure for digital identity and relationship management governed by the XNS Public Trust Organization (XNSORG).

XNS address An address that can be resolved to a node in an identity document. An identity address can be globally resolved to the root node of the identity document; an identity data address can be globally resolved to any other XNS-addressable node below the identity document root, a data address can be locally resolved to any other XNS-addressable node below the identity document root, and a relative address can be locally resolved to any node below the current node.

XNS application services Any XNS service defined as an extension to the XNS base services.

XNS base services The XNS services essential for defining and governing the technical interoperability of the XNS digital identity infrastructure, including the means by which the base services may be extended.

XNS community The group of all identities that use the XNS protocol, i.e., the schema and service definitions defined by XNSORG.

XNS contract An XNS object contained by an XNS link. An XNS contract governs the terms and conditions for the exchange of a specific set of XNS data between two or more XNS identities. An XNS link may contain any number of XNS contracts, and each contract may contain any number of XNS permissions.

XNS data Any data stored in an identity document that is an instance of a valid XNS data definition.

XNS data definition An XNS object contained in an XNS service definition that defines the schema for an XNS datatype.

XNS datatype A class of XNS data that can be instantiated once an XNS data definition exists.

XNS document See identity document.

XNS global community The community of all XNS identities that interoperate through the use of global community services such as ID and Name available from XNSORG.

XNS ID An ID that conforms to the XNS Addressing Specification. See ID.

XNS identity A digital identity that provides identity services using the XNS protocol. See Identity.

XNS link A relationship between two XNS identities represented by a special XNS datatype called a link object. A link object is the container for one or more XNS contracts. Any two XNS identities can have exactly one link.

XNS message Any identity services message that can be instantiated once an XNS message definition exists.

XNS message definition An XNS object contained in an XNS service definition. An XNS message definition defines the schema for an XNS message.

XNS message type A class of XNS messages that can be instantiated once an XNS message definition exists.

XNS name A name that conforms to the XNS Addressing Specification. See Name.

XNS node See XNS-addressable node.

XNS object Any object in an identity document, including the root node, that is derived from the XNS Core schema definition XNSObject. All XNS objects are XNS-addressable nodes.

XNS permission An XNS object contained by an XNS contract that governs the transfer and use of XNS data exchanged under that contract. XNS permissions include access control permissions, data usage (privacy) permissions, and synchronization permissions. They are extensible so they can be adapted to any form of data control between two identities.

XNS reference An XNS address contained by an XNS object that serves as a reference to another XNS object.

XNS resource Any XNS-addressable node within an identity document, including the identity document itself.

XNS service Any service that can be offered by an XNS identity once an XNS service definition exists and a service implementation is available to that identity. XNS itself is a set of services defined by the XNS Base Services Specification.

XNS service definition A special XNS datatype used as a container for XNS data definitions, XNS message definitions, XNS service dependencies, and any other data necessary to define an XNS service. XNS service definitions are defined and discovered using XNS Discovery Service.

XNS service dependency The requirement of one XNS service to have another XNS service available in order to render its service. XNS service dependencies are specified as an XNS object contained by an XNS service definition.

XNS Technical Specifications The normative technical specifications for XNS published and maintained by XNSORG.

XNS-addressable node Any XNS object contained with an identity document, i.e., any element of an identity document with an XNS ID and/or XNS Name attribute. An XNS-addressable node may contain XML elements or attributes that are not XNS-addressable, but the reverse is not true. All XNS-addressable nodes are derived from the XNS datatype XNSObject defined by the XNS Core service.

XNSObject The abstract schema definition from which the data definition of all XNS-addressable nodes in an identity document are derived. For example, the XNSObject schema defines the global attributes "ID" and "Name" for XNS IDs and XNS names. It also defines the other global attributes and elements used to address and manage identity documents and provide XNS identity services.

XNSORG The XNS Public Trust Organization-the international non-profit corporation responsible for governance of the XNS Technical Specifications and global community services. For further information see www.xns.org.

XPath The XML Path Recommendation from the W3C used for addressing inside an XML document.

XPath-addressable node Used in contrast to XNS-addressable node. Only those nodes in an identity document having XNS ID or XNS name attributes are XNS-addressable, but all XML nodes in an identity document are XPath-addressable.

XRI XNS Resource Identifier. An XNS address encoded in URI format.

APPENDIX D

References

1. R. Moats, "URN Syntax", RFC 2141, AT&T, May 1997.
2. World Wide Web Consortium, "Simple Object Access Protocol (SOAP) v1.1", W3C Note, http://www.w3.org/TR/SOAP/, 08 May 2000.
3. World Wide Web Consortium, "Web Services Description Language (WSDL) v1.1", W3C Note, http://www.w3.org/TR/wsdl, 15 March 2001.
4. World Wide Web Consortium. XML Schema Part 1: Structures, W3C Recommendation, http://www.w3.org/TR/xmlschema-1/, 2 May 2001.
5. World Wide Web Consortium. XML Schema Part 2: Datatypes, W3C Recommendation, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/datatypes.html, 2 May 2001.
6. World Wide Web Consortium. Extensible Markup Language (XML) 1.0. W3C Recommendation. http://www.w3.org/TR/1998/REC-xml-19980210. February 1998.
7. S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997
8. OASIS Security Services Technical Committee, Security Services Markup Language (SAML) Specification Set v1.0, http://www.oasis-open.org/committees/security/, 31 May 2002.
9. World Wide Web Consortium. XML Path Language (XPath) v1.0, W3C Recommendation, http://www.w3.org/TR/xpath, 16 November 1999.
10. Tim Berners-Lee, et. al, "Uniform Resource Identifiers (URI) Syntax", RFC 2396, August 1998.

APPENDIX E

XNS License

This Work (the XNS™ Technical Specifications, Technical Certification Specifications, and any associated software, documents, and other related items provided to you with notice of this License) is protected by patent, copyright, trademark, and other intellectual property rights ("IPRs") owned or licensed by the XNS Public Trust Organization ("XNSORG," www.xns.org). XNSORG provides the Work under the following license ("License"). By obtaining, using, or copying any portion of this Work, you (the "Licensee") agree that you have read and will comply with the following terms and conditions:

1. Background

1.1. XNS™ refers to "Extensible Name Service," an interoperable digital identity infrastructure based on (but not limited to) the Extensible Markup Language ("XML") family of structured data interchange standards established by the World Wide Web Consortium ("W3C") (www.w3c.org) and other Web and Internet infrastructure standards.

1.2. XNSORG is a nonprofit public trust organization that has obtained the rights to XNS™ technology and specifications in order to facilitate and promote the widespread adoption of XNS™ as an open digital identity infrastructure.

1.3. XNSORG provides the patented technology and associated specifications, documents, and other related items on an Open Standards basis, to encourage the implementation and further development of public and private XNS™ applications.

2. Definitions. All terms that are capitalized in the License, and not otherwise defined in the text, are defined in Exhibit A and incorporated herein.

3. Ownership

3.1. XNSORG has a license from OneName Corporation (www.onename.com) with respect to the Patent Rights, including the right to sublicense.

3.2. XNSORG holds the copyright and other IPRs for the XNS™ Technical Specifications, XNS™ Technical Certification Specifications, XNS™ Global Community Service Specifications, XNS™ Global Community Service Certification Specifications, and other software, documents, and related items included or referenced in this Work, except where otherwise indicated.

3.3. XNSORG owns the XNS™ service mark and certain other trademarks as indicated in the Work. Marks owned by third parties are so indicated in the Work.

4. Permission Granted by XNSORG

XNSORG grants Licensee a fully paid-up, irrevocable, royalty-free, worldwide license, on a non-exclusive basis, under the Patent Rights within the Field of Use, to make, have made, use, import, sell, offer to sell, and otherwise distribute or dispose of products and services, and also to copy, modify, and distribute the portions of this Work protected by copyright, trademark, and other Intellectual Property Rights apart from the Patent Rights, only under the following conditions:

4.1. Licensee may use the Work to develop any product intended to be distributed under an Open Source Initiative Certified License, and Licensee may make, have made, use, import, sell, offer to sell, and otherwise distribute or dispose of any product so long as it does so under an Open Source Initiative Certified License.

4.2. Licensee may develop and operate a Global Community Service in conformance with the Technical Specifications and other requirements determined by XNSORG to further the interests of the XNS Community.

4.3. Licensee may develop and operate a certification authority under agreement with XNSORG, so long as Licensee (a) complies with the Technical Certification Specifications and any applicable Global Community Service Certification Specifications and (b) promptly notifies XNSORG of new certifications.

4.4. Licensee may use the Work to develop any product intended to be a Proprietary Product, and Licensee may make, have made, use, import, sell, offer to sell, and otherwise distribute or dispose of any Proprietary Product, only under the following conditions:

4.4.1. All portions of the Proprietary Product that implement the Technical Specifications comply with the Technical Specifications and are certified pursuant to the Technical Certification Specifications in effect at the time of such certification.

4.4.2. All portions of the Proprietary Product that implement extensions of the Technical Specifications are implemented in a manner prescribed by the Technical Specifications and are certified pursuant to the Technical Certification Specifications in effect at the time of such certification.

5. Reciprocal licensing. Licensee shall grant XNSORG or its delegate a reciprocal, fully paid-up, irrevocable, royalty-free, worldwide, non-exclusive license (including the right to grant sublicenses) to make, have made, use, import, sell, offer to sell, and otherwise distribute or dispose of works patented by Licensee, and to use, copy, modify, and distribute any works covered by Licensee's copyright or other Intellectual Property Rights apart from patents, to the extent that such works are necessary to implement the Technical Specifications or conform to the Technical Certification Specifications, or to implement the Global Community Service Specifications or conform to the Global Community Service Certification Specifications with respect to a Global Community Service provided by Licensee. Licensees that grant XNSORG reciprocal licenses under this section shall be entitled to notice from XNSORG in the event that XNSORG decides not to pay maintenance fees on the Patent Rights as they come due or otherwise allow the Patent Rights to lapse.

6. Compliance with specifications. Any determinations as to compliance with the Technical Specifications shall be governed by the Technical Certification Specifications, and any determinations as to compliance with the Global Community Services Specifications shall be governed by the Global Community Services Certification Specifications.

7. Assignment and succession. This License is intended to bind successors and assigns, and XNSORG and Licensee shall take commercially reasonable steps to ensure that their respective successors and assigns are notified of their rights and obligations under this License.

8. Sublicensing. XNSORG does not grant permission to Licensee to sublicense any rights under this License to a third party, but Licensee may deliver a copy of this License in its entirety to any third party so that it can be licensed directly thereunder.

9. Notice to third parties. In the event that Licensee distributes to third parties a copy of all or a portion of this Work, separately or incorporated into another work, Licensee shall conspicuously include the following with every copy:

9.1. The IPR notices found on the Work as published by XNSORG.

9.2. The full text of this License (or a hyperlink to the full text of this License, if the copy is distributed by Internet).

9.3. Attribution, in a derivative work, of XNSORG as a source of software, documents, or other items, with a URL or hyperlink to XNSORG's website (currently found at www.xns.org).

9.4. Reference to any applicable certification by XNSORG or a certification authority with respect to the Technical Certification Specifications or the Global Community Service Certification Specifications.

10. Disclaimer of warranties and liabilities.

10.1. THIS WORK IS PROVIDED "AS IS," AND XNSORG MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE, WARRANTIES OF SYSTEM INTEGRATION, DATA ACCURACY, OR QUIET ENJOYMENT, OR WARRANTIES THAT THE USE OF THE WORK WILL NOT INFRINGE ANY THIRD-PARTY PATENTS, COPYRIGHTS, TRADEMARKS, OR OTHER RIGHTS.

10.2. XNSORG WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF OR RELIANCE ON THE WORK.

11. Trademark and publicity. Apart from the notices required herein, Licensee shall not use the XNSORG name and trademarks in advertising or publicity without specific, written prior permission.

12. Indemnification. Licensee shall defend, indemnify and hold XNSORG harmless from and against any and all actions, suits, or proceedings ("Claims") resulting in any losses, damages, judgments, awards, settlements, or expenses (including reasonable attorney's fees) (collectively, "Liabilities") arising from any breach of Licensee's obligations under this License.

13. Term. The license granted herein shall commence when licensee first obtains this Work and thereafter continue in each jurisdiction throughout the world for the statutory duration of the Patent Rights or other IPRs covered by this License in any such jurisdiction, unless earlier terminated in accordance with the terms hereof.

14. Termination. In addition to any other available legal or equitable remedies, XNSORG may terminate the license granted herein if Licensee is in breach or default of any of the material terms and conditions of this Agreement, provided that Licensee shall not have cured (if capable of being cured) such breach or default within thirty (30) cumulative days after receipt of such notice. Please note that under the terms of XNSORG's license from OneName under the Patent Rights, any termination of XNSORG's license shall not affect XNSORG's sublicenses, which are to be automatically assigned to OneName or to a non-profit organization operating for the benefit of the XNS Community.

15. General Terms.

15.1. *Governing Law:* This License shall be construed and controlled by the laws of the State of Washington without reference to conflict of laws principles.

15.2. *Jurisdiction:* The parties agree that all disputes arising in any way out of this License shall be heard exclusively in, and all parties irrevocably consent to jurisdiction and venue in, the United States District Court for the Western District of Washington or in King County Superior Court.

15.3. *Not Partners:* The parties hereto are independent entities and are not partners or joint venturers with each other.

Exhibits: The following Exhibits are attached hereto and incorporated by reference: Exhibit A (Definitions), Exhibit B (Original Base Services), and Exhibit C (Open Source Initiative License Description)

EXHIBIT A - DEFINITIONS

1. Addressing Specification - The portion of the Technical Specifications that defines the syntax and normalization rules for the federated identifiers, names, and addresses used in ID Service and Name Service and the rules for using these addresses to validate identity documents and document fragments.

2. Base Services - The set of services defined in the Base Services Specification.

3. Base Services Specification - The portion of the Technical Specifications that defines the set of schemas and services essential for defining and governing the technical interoperability of XNS, including the means by which the Base Services may be extended.

4. Field of Use - The development and exploitation of computer programs that enable data communication across the Internet and private and public data networks and that are developed pursuant to the Technical Specifications as first delivered and/or subsequently modified.

5. GCSP (Global Community Service Provider) - A licensed subcontractor of XNSORG that provides Global Community Services.

6. General Identity - An identity that is not a Personal Identity or Organizational Identity, i.e., any generic object, entity, category, or taxonomy entry whose identity is defined by linguistic, cultural, or scientific convention.

7. General Vocabulary - The generic identities, identity names registered in General Name Service, schema definitions, taxonomies, and ontologies established and maintained by XNSORG as a public service available to all members of the XNS Community to enhance the utility and interoperability of the XNS digital identity infrastructure and to enable global vocabulary sharing and reuse.

8. Global Community Service Specifications - The supplemental Technical Specifications governing the operation of each Global Community Service, to be initially provided to XNSORG, in each case, by the primary GCSP for the applicable Global Community Service prior to the commencement of such Global Community Service.

9. Global Community Services - From among the XNS services defined in the Technical Specifications, the subset of XNS services offered by XNSORG to all members of the XNS Community to facilitate interoperability of the XNS digital identity infrastructure and that shall be supported in all implementations developed pursuant to the Technical Specifications.

10. Global Community Services Certification Specifications - Test suites, conformance criteria, trust marks, or other requirements which enable certification authorities to certify, and the XNS Community and the public to determine, conformance by GCSPs, Registrars, and Registrants to the Global Community Service Specifications published by XNSORG.

11. Identity - An object, entity, or other encapsulation of related attributes that can be uniquely addressed and is capable of independently participating in relationships with other such objects.

12. Identity Document - An XML document (or the equivalent) encapsulating the tree of attributes associated with an identity at a particular network location.

13. Identity Link - A relationship between two identities encapsulating one or more agreements to transact identity documents or identity attributes.

14. Identity Service - The set of operations associated with an identity document that performs actions on behalf of the Principal.

15. Identity Service Provider - The legal entity responsible for operation of an identity service.

16. Intellectual Property Rights - All rights in patent, trademark, copyright, trade secret and know-how.

17. Open Source Initiative Certified License - A license certified by the Open Source Initiative (www.opensource.org) and listed on the XNSORG website (www.xns.org). (The list of certified licenses as of July 9 2002 is included in Exhibit C.)

18. Open Standard – As defined on July 9, 2002, at http://www.perens.com/OpenStandards/Definition.html, a specification whose licensing ensures that it is publicly available to implement, that it maximizes end-user choice of vendors and implementations, that it does not require a royalty or fee to implement with the potential exception of compliance certification, that it does not discriminate against vendors or implementations, and that it permits extensions or subsets that are consistent with the standard yet may preclude predatory "embrace and extend" practices.

19. Organizational Identity - An identity representing any legal entity that is not a personal identity.

20. Original Base Services - The set of services defined in Exhibit B.

21. Original Global Community Services – The following subset of the Original Base Services: Core, Discovery, ID, Session, Personal Name, Organization Name, General Name, Personal Directory, Organization Directory, General Directory, Personal Reputation, Organization Reputation, and General Reputation.

22. Patent Rights - (1) United States letters of patent issued under numbers 5,862,325; 6,044,205; 6,088,717 and 6,345,288; (2) any and all United States or foreign letters of patent, utility models and/or applications therefore, claiming priority, in whole or in part, from any of the letters of patent identified herein in (1) of this definition; (3) any and all divisionals, continuations, continuations in part, continued prosecution applications, reexaminations, reissues, additional or extension of any of the letters of patent or utility models identified herein in (1) and (2) of this definition.

23. Personal Identity - An identity representing a natural-born person.

24. Principal - The real-world entity or concept represented by an identity document.

25. Proprietary Products - Products that are not distributed under an Open Source Initiative Certified License.

26. Registrant - A member (organization or individual) of the XNS Community that enrolls with a Registrar or XNS service provider to obtain Global Community Services.

27. Registrars - A GCSP-registered service provider responsible for enrolling Registrants to obtain Global Community Services.

28. Technical Certification Specifications - Test suites, conformance criteria, trust marks, or other requirements that enable certification authorities to certify, and the XNS Community and the public to determine, conformance by implementers to the Technical Specifications published by XNSORG or its delegate.

29. Technical Specifications - The technical specifications for XNS digital identity infrastructure, including the Base Services Specification and the Addressing Specification, as originally submitted by OneName to XNSORG and subsequently modified, amended, revised or enhanced, and published on the XNSORG website.

EXHIBIT B – ORIGINAL BASE SERVICES

1. ID Service – a service for registering and resolving federated persistent identifiers for an Identity optimized for machine resolution according to the addressing syntax defined in the Addressing Specification.

2. Name Service – a service for registering and resolving federated semantic identifiers for an Identity optimized for human usability and memorability according to the addressing syntax defined in the Addressing Specification.

    a. Personal Name Service – Name Service provided for Personal Identities.

    b. Organizational Name Service – Name Service provided for Organizational Identities.

    c. General Name Service – Name Service provided for General Identities.

3. Core Service – a service for defining the core abstract and concrete schemas from which all other XNS schemas and services are derived.

4. Discovery Service – a service for defining and discovering the schemas and services available from an Identity.

5. Hosting Service – a service for creating, deleting, and moving Identity Documents.

6. Data Service – a service for creating, reading, updating, and deleting identity attributes from Identity Documents.

7. Folder Service – a service for managing, organizing, and searching identity attributes within an Identity Document.

8. Directory Service – a service for managing, organizing, and searching Identity Documents.

    a. Personal Directory Service – Directory Service provided for Personal Identities.

    b. Organizational Directory Service – Directory Service provided for Organizational Identities.

    c. General Directory Service – Directory Service provided for General Identities.

9. Authentication Service – a service for validating the credentials of a Principal to be the controller of an Identity Document.

10. Session – a service for validating that a Principal has authenticated their identity at an Identity Service Provider.

11. Certification – a service for obtaining a credential asserting the validity of one or more identity attributes.

12. Reputation – a service for obtaining a credential asserting a value judgment about an Identity or attributes of an Identity.

    a. Personal Reputation Service – Reputation Service provided for Personal Identities.

    b. Organizational Reputation Service – Reputation Service provided for Organizational Identities.

    c. General Name Reputation – Reputation Service provided for General Identities.

13. Negotiation – a service for agreeing to the terms and conditions under which an Identity Link shall be created, maintained, or terminated.

14. Introduction – a service by which an Identity linked to other identities can negotiate an Identity Link between those other Identities.

EXHIBIT C – OPEN SOURCE INITIATIVE LICENSE DESCRIPTION

The following is a list of the Open Source Initiative Certified Licenses effective July 9, 2002. See the XNSORG website (www.xns.org) for an updated list of certified licenses and effective versions.

Apache Software License
Apple Public Source License
Artistic license
BSD license
Common Public License
Eiffel Forum License
GNU General Public License (GPL)
GNU Library or "Lesser" Public License (LGPL)
IBM Public License
Intel Open Source License
Jabber Open Source License
MIT license
MITRE Collaborative Virtual Workspace License (CVW License)
Motosoto License
Mozilla Public License 1.0 (MPL)
Mozilla Public License 1.1 (MPL)
Nethack General Public License
Nokia Open Source License
Open Group Test Suite License
Python license (CNRI Python License)
Python Software Foundation License
Qt Public License (QPL)
Ricoh Source Code Public License
Sleepycat License
Sun Industry Standards Source License (SISSL)
Sun Public License
University of Illinois/NCSA Open Source License
Vovida Software License v. 1.0
W3C License
X.Net License
Zope Public License
zlib/libpng license