XNS Addressing Specification v1.1

26 March 2003

Editors

Dave McAlpin, Epok Inc., dave.mcalpin@epok.com

Drummond Reed, OneName Corporation, drummond.reed@onename.com

Contributors

Mike Lindelsee, Visa International, mlindels@visa.com

Gabe Wachob, Visa International, gwachob@visa.com

Loren West, Epok Inc., loren.west@epok.com

Document History

Version #	Date	Editor	Scope of changes
01	06-Sep-2002	Dave McAlpin	Initial publication on XNSORG wiki
02	23-Mar-2003	Dave McAlpin	V02 draft
03	26-Mar-2003	Drummond Reed	V03 draft

Table of Contents

1 About the XNS Public Trust Organization (XNSORG)

XNSORG is a non-profit organization created to develop and manage Extensible Name Service (XNS) in the public interest. XNS is an open, XML-based addressing system and data exchange protocol for identifying and linking any resource participating in any kind of digital transaction. The complete technical specificatons for XNS are available at the XNSORG website at http://www.xns.org.

XNSORG has been granted intellectual property rights by contributors to the XNS specifications, and in turn makes these rights available to the public under an open, royalty-free license as described in http://www.xns.org/pages/XNS_License.pdf.

2 About this Document

This document is the formal specification for XNS Addressing Specification v1.1. It is a minor update to the XNS Addressing Specification portion of the XNS Technical Specifications v1.0, available in their entirety at http://www.xns.org/pages/XNS_Technical_Specs.pdf.

Since the abstract addressing concepts in XNS are useful outside of the scope of XNS itself, this specification is being published as a standalone document so it can be referenced independently from the full XNS specifications.

3 Terminology and Conventions

The following conventions are used in this document:

- 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 [1].
- EBNF productions use the EBNF syntax notation as described in the W3C XML 1.0 Recommendation [2].

4 Introduction to XNS Addressing

The W3C XPath 1.0 Recommendation [3] establishes a standard syntax for addressing the nodes of a structured XML document. Since XNS Addressing provides addresses for a network of linked XML documents, it has a similar need for a standardized syntax.

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 must 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 rules for XNS address validity. It includes four rule sets:

1. The EBNF definition of XNS addressing syntax.
2. The EBNF definition for URI encoding of XNS addresses and XNS service invocations.
3. XNS ID normalization rules.
4. XNS Name normalization rules.

5 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 [2] 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 = ':' [HostIDNode *('.' HostIDNode)] ':' [IdentityIDNode *('.'
IdentityIDNode)]
[ 4] HostIDNode ::= INT | ('(' IdentityID | URN ')')
[ 5] INT ::= Non-negative integer
[ 6] URN ::= Uniform Resource Name as specified in IETF RFC 2141 [5]
[ 7] IdentityIDNode ::= ID | ('(' IdentityID | URN ')')
[ 8] ID ::= XML character string normalized according to the XNS ID Normalization
Rules
[ 9] IdentityDataID ::= IdentityID DataID
[10] DataID ::= ';' DataIDNode [RelativeDataID]
[11] DataIDNode ::= ID | ('(' (IdentityDataID  | URN) ')')
[12] RelativeDataID ::= *('.' DataIDNode) [',' Version]
[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 | URI ')')
[20] Name ::= XML character string normalized according to the XNS Name
Normalization Rules
[21] IdentityAddress ::= (IdentityID ['!' IdentityName]) | IdentityName
[22] URI ::= Uniform Resource Identifier as specified in IETF RFC 2396 [4]
[23] IdentityDataName ::= IdentityName DataName
[24] DataName ::= '/' RelativeDataName
[25] RelativeDataName ::= DataNameNode *('/' DataNameNode) ['/,' Version]
[26] DataNameNode  ::= Name | ('(' (IdentityDataAddress | URI) ')')
[27] IdentityDataAddress ::= (IdentityDataID ['!' IdentityDataName]) |
IdentityDataName
[28] AbsoluteAddress ::= IdentityAddress | IdentityDataAddress
```

5.1 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.

5.1.1 IDs, Names and Addresses

Three of the most fundamental requirements of XNS addressing are the ability to:

1. Provide an abstraction layer capable of representing the identity of any network actor or entity—machine, network location, application, user, business, taxonomy category, etc.,
2. Enable this identity to persist for the lifetime of the resource it represents, and
3. Enable this identity abstraction layer to be federated across any number of communities for fully decentralized, delegated identity management.

To meet these requirements XNS addressing 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 addressing a name is normally resolved 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) [4]. Since URIs do not require persistence of the address, an XNS ID meets the higher persistence requirements of a Uniform Resource Name (URN) [5].

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

- **IDs** are persistent 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. An ID is a URN, i.e., it may expire, but it may never be assigned to another identity or identity attribute. Likewise, 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.

- **Names** are non-persistent addressing values intended primarily for human use. XNS names typically 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 usually resolves to an XNS ID before it is resolved to the network location of the target, e.g., a URI.

- **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.

5.1.2 Object Versions

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 identity attributes at different moments in time. To unambiguously address a specific version of an identity attribute, the EBNF "Version" productions allow the version value identifying the target version 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 identity 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 a dateTime instance (as specified by W3C XML Schemas Part 2 [6]). See the *Version Date Format* rule for more about this format.

5.1.3 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 addressing 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 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 representation of that top- level container. All nodes below the root node of the identity document represent the attributes (data) of this container.

Since all XNS identities are absolute, they require absolute addresses, and registering and resolving these addresses may require inter-identity communications because the addresses may 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.

5.1.4 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 may be represented by a resource that physically resides somewhere on the network, an XNS identity address, if resolvable, must resolve to the network address of this resource.

In XNS, a special type of identity called a *host identity* represents the identity of a network endpoint—a device with a physical network address at which an identity may be contacted. A host identity is simply an XNS identity with at least one known set of attributes: a list of URIs over which this host machine accepts 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 point out 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 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.

5.1.5 Cross-references

The final key concept in XNS addressing architecture is one that is critical to a distributed, federated identity system. It is also one of the most novel aspects of XNS addressing. There are times it is necessary to refer to an identity in a particualar namespace by the address by which that identity is known in a different namespace. For example, imagine that Alice has a corporate email address of alice@abc.com. Alice changes jobs and gets a new email address of alice.smith@newco.com. Bob knows Alice's old address and he knows she now works at newco.com, but he doesn't know her new address. Alice's new company, newco.com, knows Alice's old address. It would be nice if Bob could somehow send mail to "that identity at newco.com that is known as alice at abc.com." In other words, Bob would like to address an identity in the newco.com namespace in terms of that identity's address in the abc.com namespace. XNS addressing provides this type of cross-community addressing via a feature called *cross-references*.

Cross-references are supported syntically by enclosing them in parentheses. The value inside the parentheses is either a fully qualified XNS Address of a fully qualified URI or URN. For example, the mailto URI scheme does not support cross-references, but if it did, a mailto address that incorporated the cross-reference described above might look something like

```
mailto:(mailto:alice@abc.com)@newco.com
```

Obviously this would only work if newco.com could make sense of the cross-reference. That is, newco.com would need some way to map the cross-reference (mailto:alice@a.com) to alice_smith, Alice's name in the newco.com namespace.

Broadly then, XNS addressing provides the ability to express logical equivalence so the same identity or identity attribute can be recognized across multiple host communities. (Note that this behavior is not required, and in fact may be expressly prohibited when an identity controller wishes to remain anonymous or pseudonymous). In XNS any element of an identity document, including the document root object representing the identity itself, can be cross-referenced with another logically equivalent element in a different identity document. Furthermore, XNS addressing also allows any URN to be used as a cross-reference for an XNS ID, and any URI to be used as a cross-reference for an XNS name.

To support the ability to make cross-references, 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 node in an XNS ID path that terminates in the root node of an identity document. IdentityIDNodes can be addressed by either a local ID, a cross-reference ID, or a URN.

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

5.2 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.

5.2.1 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.

5.2.2 XNS IDs

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

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

5.2.3 Identity IDs

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

An IdentityID is a fully formed address to an identity (or to the root node of the abstract document representing an identity). An IdentityID consists of a path that begins with a colon representing the abstract XNS ID community identity. This is followed by zero or more dot separated HostIDNodes that, taken together, form a globally unique value representing the host identity. Line 4 defines a HostIDNode as either 1) an integer or 2) a cross-reference containing an IdentityID or a URN. Note that it is possible to have no HostIDNodes, in which case the host is the XNS ID community identity.

If the HostIDNodes in an IdentityID are all integers (i.e. the host identity is expressed without cross-references), the host identity is in the form of an Object ID (OID). An OID 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. In the case of XNS, the OID root is the XNS ID community identity managed by XNSORG on behalf of the global community.

If a host is identified by a different addressing community, it must have a cross-reference for its first-level HostIDNode. To meet the persistence requirements of an XNS ID, this cross-reference must contain either: a) the IdentityID of an host identity defined in a different XNS addressing community, or b) a URN as defined in IETF RFC 2141 [5].. Like all cross-references, the IdentityID or URN is enclosed in parentheses. One common form of a URN that works well for a peer-to-peer XNS addressing community is 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).

Following are two examples of IdentityIDs in which the root HostIDNodes are cross-references expressed as URNs. The first is a UUID-based URN. The second is a URN system called Handle operated by the Corporation for National Research Initiatives (CNRI) [7].

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

Because they are native to XNS, OID-based IdentityIDs tend to be 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)
```

Lines 3 and 4 allow any node in the host identity to be a cross-reference. For example, a cross-reference containing a URN could be the top-level host identity node and OIDs could be used for lower-level nodes. Examples:

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

To address any identity inside a host community, the host IdentityID is followed by a second colon followed by the hosted identity. The hosted identity is a path of one or more IdentityIDNodes delimited by dots. Note that lines 7 and 8 specify that the IdentityIDNode 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
```

IdentityIDNodes, like HostIDNodes, may be expressed as cross-references. This allows, for example, 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, all cross-references are 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)
```

5.2.4 Identity Data IDs

```
[ 9] 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
```

5.2.5 Data IDs

```
[10] 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 element. 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      (legal but not advised)
;email.homePhone   (legal but not advised)
```

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 because XNS IDs, like all URNs, must continue to reference the same resource in spite of changing semantic relationships.

5.2.6 Relative Data IDs

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

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   (legal but not advised)
```

Line 11 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)
```

5.2.7 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 attribute associated with the data node. 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
```

5.2.8 XNS Names

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

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

5.2.9 Identity Names

```
[17] IdentityName ::= (NamespaceSymbol | '//') IdentityNameNode
*('/' IdentityNameNode)
[18] NamespaceSymbol ::= '=' | '@' | '+'
[19] IdentityNameNode ::= Name | ('(' IdentityAddress | URI ')')
[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.)

2. **Identity names can contain cross-references** to other identities (line 19). This capability is very useful in federated identity management. The cross-reference can be expressed as either an IdentityAddress or a URI. (See examples below.)

3. **Namestrings can be any legal XML characters** as defined by the W3C XML 1.0 Recommendation [2], 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 naming root. 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 the community rooted on //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
//xns/per/John Smith
=John Smith, Jr.
//xns/per/John Smith, Jr.
```

Examples of XNS organizational identity names:

```
@Example
@Example/Computers
@Example/Computers/Internet
@Smith & Jones
@John Smith Inc.
//xns/org/John Smith, Inc.
```

Note that in the second and 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
//xns/gen/Dominican Republic
```

Examples of identity names using international character sets:

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

Examples of identity names using IdentityAddresses as cross-references:

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

Example of identity names using a URI:

```
//(mailto:john.smith@example.com)/data/tel/Work
```

5.2.10 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 requiring resolution.

If an IdentityID is not present, then an IdentityAddress must contain an IdentityName, which can then 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.

5.2.11 URI

```
[22] URI ::= Uniform Resource Identifier as specified in IETF RFC 2396 [4]
```

A URI according to RFC 2396 [4]. The full BNF is available in that document. Being able to use a URI as a cross-reference is one of the most powerful features of XNS Addressing, as it permits the XNS identity of any resource with an existing URI today to be located using that URI.

5.2.12 Identity Data Names

```
[23] 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.

5.2.13 Data Names

```
[24] 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
```

5.2.14 Relative Data Names

```
[25] RelativeDataName ::= DataNameNode *('/' DataNameNode) ['/,' Version]
[26] DataNameNode  ::= Name | ('(' (IdentityDataAddress | URI) ')')
```

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 provide the same versioning capability as XNS IDs, the same versioning syntax can be appended to an XNS data name 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 26 specifies that data names can also incorporate cross-references which themselves can be either IdentityDataAddresses (below) or URIs. Examples:

```
@Yahoo/Computers/Internet/(@IBM/Computers/AS400)
@Smith & Reilly/Tools/(+garden rakes/price)
=John Smith/Friends/(=Mary Frank/Tel/Home)
=John Smith/Friends/(mailto:mary.frank@example.com)/Tel/Home
=John Smith/Friends/(http://www.maryfrank.com)/Tel/Home
```

5.2.15 Identity Data Addresses

```
[27] 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 be 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 can 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
```

5.2.16 Absolute Addresses

```
[28] 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.

6 XRI (XNS Resource Identifier) EBNF Definition

To be useable on the Web, an XNS address must first be specified in URI 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 [4], it MUST conform to the following EBNF definition.

```
[01] XRI ::= URIScheme (ServiceCall | IdentityAddress | IdentityDataAddress)
[02] URIScheme ::= HTTP | URN | XNS
[03] HTTP ::= ['http://' | 'https://'] 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 ::= ArgName '=' ArgValue
[10] ArgName ::= The argument name as defined by the message specification
[11] ArgValue ::= The value of the argument as a string, with ';', ')', and '\'
characters escaped with a '\'
[12] IdentityAddress ::= as defined in XNS Address EBNF
[13] IdentityDataAddress ::= as defined in XNS Address EBNF
```

The three URI prefixes correspond to the three URI schemes [4] that will most commonly be used with XNS addresses:

- **The HTTP scheme** is used to direct 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 direct XNS address resolution requests to a URN resolver. NOTE: only XNS addresses that consist entirely of IdentityIDs or IdentityDataIDs qualify as URNs due to the persistence requirements of URNs [5].
- **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.

7 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. XNSORG expects to publish a formal EBNF definition for XNS ID normalization at a future date. Until then, the following top-level rules are normative:

7.1 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 [2].

7.2 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.

7.3 XNS Global Community ID

All XNS ID resolvers SHOULD be able to internally resolve the following XNS Global Community ID:

```
"::" - resolves to the following list of URIs:
 http://resolver.xns.org/xns
 https://resolver.xns.org/xns
```

In addition, XNS ID resolvers SHOULD be able to internally resolve the following IDs:

```
":1:" - resolves to the following list of URIs:
  http://per.xns.org/xns
  https://per.xns.org/xns

":2:" - resolves to the following list of URIs:
  http://org.xns.org/xns
  https://org.xns.org/xns

":3:" - resolves to the following list of URIs:
  http://gen.xns.org/xns
  https://gen.xns.org/xns
```

8 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. XNSORG expects to publish a formal EBNF definition for XNS Name normalization. Until then, the following top-level rules are normative:

8.1 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 [2].

8.2 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.

8.3 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 [2].

8.4 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 be allowed by the XNS name normalization rules, including this character itself.

8.5 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.

8.6 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.

9 References

1. S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997.
2. World Wide Web Consortium. Extensible Markup Language (XML) 1.0. W3C Recommendation. http://www.w3.org/TR/1998/REC-xml-19980210. February 1998.
3. World Wide Web Consortium. XML Path Language (XPath) v1.0, W3C Recommendation, http://www.w3.org/TR/xpath, 16 November 1999.
4. Tim Berners-Lee, et. al. “Uniform Resource Identifiers (URI) Syntax”, RFC 2396, August 1998.
5. R. Moats, “URN Syntax”, RFC 2141, AT&T, May 1997.
6. 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.
7. Sam X. Sun, et. al. Handle System Namespace and Service Definition, http://www.ietf.org/internet-drafts/draft-sun-handle-system-def-07.txt, November 2002.

