Metadata description and discovery protocols

Version: 1.0-06

Editors:
Peter Davis, NeuStar, Inc.

Contributors:
Paul Madsen, Entrust, Inc.
John Beatty, Sun Microsystems, Inc.
David McCalpin, OneName
Jeff Hodges, Sun Microsystems, Inc.
Bronislav Kavsan, RSA Security, Inc.
Scott Cantor, Internet2

Abstract:

This document details metadata, protocols for obtaining metadata, and methods of resolution for discovering the location of metadata for the Liberty Identity Services Framework

Notice

Liberty Alliance Project
Licensing Administrator
c/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08855-1331, USA
info@projectliberty.org

Revision History

Revision: 05 **Date:** 03 April

- Bugref: 209 - Relaxed TLS URI restriction for publication URI's (/MUST/SHOULD/)
- Bugref: 118 - Added protocol version support as attribute on prodiverDesriptorType
- Bugref: 204 - Added clarity to the affiliationID uniqueness requirement
- Bugref: nil - Minor organizational changes, syntax errors, etc...

Revision: 06 **Date:** 13 April

Miscellaneous changes; Closing bugs 119, 208, 209

Contents

1. Introduction 5
    1.1. Notation and Conventions 5
    1.2. Overview 5
2. Metadata Schema 5
    2.1. Entity Descriptors 6
    2.2. Schema Declarations 6
    2.3. Complete Metadata Schema 16
3. Publishing the Metadata 18
    3.1. Using the DNS to publish metadata location(s) 19
    3.2. Publication via Well-Known Location 21
4. Metadata Resolution and Retrieval 21
    4.1. Resolving Locations and Retrieving Metadata 22
5. Post Processing of the Metadata document 23
    5.1. Processing of ds:Signature and general trust processing 24
    5.2. Metadata Location and Document Caching 24
    5.3. Handling of HTTPS Redirects 25
6. Security Considerations 25
    6.1. Trust Establishment 26
7. References 26

1. Introduction

Within the version 1.1 and 1.2 *ID-FF* specifications of the Liberty Alliance protocols, basic metadata were exchanged out-of-band between entities. With this release, which more formally describes metadata, schema and protocols are introduced to facilitate real-time requests for this data, in order to speed implementation, and allow for more spontaneous conversations between Liberty-compliant entities.

Functionally, there are three primary classes of metadata:

- entity core metadata, which covers the metadata elements introduced in release 1.0 of the protocol, with additional elements introduced in this release, including expression of multiple entities, and additional entities not previously described
- entity trust metadata, which enables entities to cast business decisions based on the characteristic trust information provided in this class, conveyed through document signature(s), DNS zone signatures, and optionally additional material that publishers may convey within the `Extension` and `AdditionalMetaLocation` elements
- origin and document verification through signature use in (server authenticated) HTTPS sessions, DNS signatures, and document level signatures

This document presents extensions to the model for metadata described in Liberty Release 1 to better support ad hoc interactions between entities. The location of cryptographic keys in a distributed-computing architecture that contains "arms-length" peer domains presents an opportunity for some fresh thinking. Conventional solutions to this problem fail to fully exploit the potential of the evolving Web Services architecture to minimize administrative costs. Liberty Phase 2 operations between previously unintroduced parties will benefit from any mechanisms that simplify how keying material can be discovered, leading to mechanisms for trust establishment and services invocations.

1.1. Notation and Conventions

This specification uses schema documents conforming to W3C XML Schema (see [Schema1]) (*TODO:* need proper reference here) and normative text to describe the syntax and semantics of XML-encoded messages.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

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

The following namespaces are referred to in this document:

1.2. Overview

The metadata protocols specified in this document will enable two Liberty-enabled entities to exchange or request cryptographic keys and service information metadata in real time, allowing dynamic interactions between these parties, eliminating the need for out-of-band negotiations between these parties a-priori. The addition of interactions between separate authentication authorities and identity chaining in the Liberty *ID-WSF* will depend upon this exchange, as portions of a principle's identity may be previously established outside the range of providers established agreements.

2. Metadata Schema

The metadata schema allows for several methods of representation:

- A single document for multiple entities (separate `providerID`'s)
- A single document for a single entity (one `providerID`)
- Multiple documents (at different URIs) which individually describe portions of the entity's federated identity services architecture (involving one or more `AdditionalMetaLocation` elements)

If separate documents are used, references to each MUST be made, either through one or more additional PID2MD NAPTR record(s), or using the `AdditionalMetaLocation` element within a document which has an associated NAPTR RR, or which is situated at the "well-known location" (see Section 3.2).

2.1. Entity Descriptors

The metadata schema consists of two primary forms and an alternate form:

- A single document expressing all of the metadata for a single entity identified by one or more `providerID` identifiers
- A single document describing multiple entities identified by multiple `providerID` identifiers
- Documents which reside at more than one location, whose locations are described either by multiple NAPTR resource records, or through the use of the `AdditionalMetaLocation` element

2.2. Schema Declarations

The metadata schema is constructed in such a way to allow an entity, described by one or more `providerID`'s publish single or multiple schema instances to describe their identity architecture services.

The primary container for a published document is either `EntityDescriptor` or the plural form `EntitiesDescriptor` (used when an affiliated set of entities chooses to publish a consolidated set of metadata documents as one).

The immediate child nodes of `EntityDesciptor` expects one or more of:

- `SPDescriptor`
- `IDPDescriptor`
- `ServiceMDDescriptor`
- `AffiliationDescriptor`

which are described below. Additionally, an extension point `Extension` is provided in order to convey additional metadata.

2.2.1. Namespaces in metadata

The following namespace declarations are used to complete the metadata schema:

- `isf:` is described by the *Liberty Identity Services Framework* schema
- `libMD:` is described by the Liberty Metadata schema (this documents schema declarations)
- `ds:` is described by the W3C XML Signature schema
- `pip:` is described by the Liberty Personal Information Profile schema, and is used to describe contact persons cited in instance documents
- `saml:` is described by the *SAML* Assertion Specification

Schema Fragment:

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
   schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
<xsd:import namespace="urn:liberty:isf:1.0" schemaLocation="draft-sun-isf-14.xsd"/>
<xsd:import namespace="urn:liberty:pip:1.0" schemaLocation="lib-svc-id-pp.xsd"/>
<xsd:import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
   schemaLocation="http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-assertion-01.xsd"/>
```

2.2.2. datatype `entityIDType`

The datatype `entityIDType` restricts the XML data to a length of 1024 bytes.

Additionally, the `entityIDType` structure is defined by the following BNF, derived from *URI*:

```
 BNF for Liberty entityIdentifiers
    # constraint on absoluteURI
    entityID     = absoluteURI [ "#" fragment ]
    absoluteURI  = scheme ":" ( hier_part | opaque_part )

    # constraint on hier_part (net_path only)
    hier_part    = net_path [ "?" query ]
    opaque_part  = uric_no_slash *uric

    uric_no_slash = unreserved | escaped | ";" | "?" | ":" | "@" |
                "&" | "=" | "+" | "$" | ","

    net_path     = "//" authority [ abs_path ]
    abs_path     = "/"  path_segments

    ; pragmatically, scheme SHOULD be an officially IANA registered URI scheme
    ; http://www.iana.org/assignments/uri-schemes
    scheme      = alpha *( alpha | digit | "+" | "-" | "." )

    authority    = server | reg_name

    reg_name     = 1*( unreserved | escaped | "$" | "," |
                   ";" | ":" | "@" | "&" | "=" | "+" )

    server      = [ [ userinfo "@" ] hostport ]
    userinfo     = *( unreserved | escaped |
                  ";" | ":" | "&" | "=" | "+" | "$" | "," )

    hostport     = host [ ":" port ]
    ; constraint on host (no ipAddress)
    host        = hostname
```

```
    hostname     = *( domainlabel "." ) toplabel [ "." ]
    domainlabel  = alphanum | alphanum *( alphanum | "-" ) alphanum
    toplabel     = alpha | alpha *( alphanum | "-" ) alphanum
    port        = *digit

    path        = [ abs_path | opaque_part ]
    path_segments = segment *( "/" segment )
    segment      = *pchar *( ";" param )
    param       = *pchar
    pchar       = unreserved | escaped |
                  ":" | "@" | "&" | "=" | "+" | "$" | ","

    query       = *uric

    fragment     = *uric

    uric        = reserved | unreserved | escaped
    reserved     = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
                  "$" | ","
    unreserved    = alphanum | mark
    mark        = "-" | "_" | "." | "!" | "~" | "*" | "'" |
                  "(" | ")"

    escaped      = "%" hex hex
    hex         = digit | "A" | "B" | "C" | "D" | "E" | "F" |
                        "a" | "b" | "c" | "d" | "e" | "f"

    alphanum     = alpha | digit
    alpha       = lowalpha | upalpha

    lowalpha = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
           "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
           "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
    upalpha  = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
           "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
           "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
    digit   = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
           "8" | "9"
```

The schema fragment for `entityIDType`:

```
<xsd:simpleType name="entityIDType">
   <xsd:restriction base="xsd:anyURI">
      <xsd:maxLength id="maxlengid" value="1024"/>
      <xsd:pattern value=""/>
   </xsd:restriction>
</xsd:simpleType>
```

2.2.3. Common Attributes

Several common attributes are defined and generally used throughout the schema:

- `providerID` of type `entityIDType` indicates the providerID of the entity described by the children of the node
- `validUntil` of type `dateTime` indicates the expiration date and time of the node (and it's descendants). If `dateTime` expressions evaluate to nonequivalent values, parsers MUST adhere to the most restrictive value (the earliest dateTime).
- `cacheDuration` of type `duration` indicates the maximum elapsed time a consumer may cache the metadata document (or fragment). Consistent with the `validUntil` attribute, the most restrictive value MUST be used when conflicting cache directives occur

Publishers MUST provide either a `validUntil` or `cacheDuration` attribute when publishing metadata. Since this directive is available at both the top-level EntityDescriptor and it immediate descendants, care should taken in selecting expiration settings. It is RECOMMENDED that publishers express document expiration at the `EntityDescriptor` element only, and not on the child nodes.

If consumers send an *HTTP (1.1)* request to the publisher URL with a header If-Modified-Since: [last retrieval dateTime] , the publisher server returns a *304 Not-Modified* response, and the publisher expresses the expiration as a `cacheDuration`, the consumer MAY reset the retrieval dateTime, effectively reseting the duration clock (see Section 5.2).

The schema fragment for the common attribute:

```
<xsd:attribute name="libertyPrincipalIdentifier" type="entityIDType"/>
<xsd:attribute name="providerID" type="libMD:entiyIDType"/>
<xsd:attribute name="validUntil" type="xsd:dateTime"/>
<xsd:attribute name="cacheDuration" type="xsd:duration"/>
```

2.2.4. Common Elements

There are several common elements defined globally, and used throughout the schema:

2.2.4.1. `Organization` element

The `Organization` element provides some basic information consumers may require when interacting with a principal:

- `OrganizationName` of type `string`: the Organizational Name of the entity, generally the complete Organization Legal name
- `OrganizationDisplayName` of type `string`: an organization name suitable for display to a principal
- `OrganizationURL` of type `anyURI`: a URL of the organization suitable for dereferencing by a user-agent, which may be used for directing a principal for additional information on the entity

```
<xsd:element name="Organization" type="organizationType"/>
<xsd:complexType name="organizationType">
   <xsd:sequence>
      <xsd:element name="OrganizationName" type="xsd:string"/>
      <xsd:element name="OrganizationDisplayName" type="xsd:string"/>
      <xsd:element name="OrganizationURL" type="xsd:anyURI"/>
   </xsd:sequence>
</xsd:complexType>
```

2.2.4.2. `ContactPerson` element

The ContactPerson element conveys general contact information for human-to-human contact regarding an entity. It is cast as a type pip:PIPType *PIP*, with the the following attribute extensions:

- `libertyPrincipalIdentifier` [optional] : a principals dereferencable nameIdentifier of type entityIDType which may point to an online instance of the person's PIP profile
- `contactType` [optional] : the type of contact, which may be one of `technical`, `administrative`, `billing`, or `other`. The default value is `technical`

The schema fragment for the common elements:

```
  <xs:element name="ContactPerson" type="contactType"/>
  <xs:complexType name="contactType">
     <xs:complexContent>
        <xs:extension base="pip:PIPType">
           <xs:attribute ref="libertyPrincipalIdentifier" use="optional"/>
           <xs:attribute name="contactType" default="technical" use="required" type="attr.contactType"/>
        </xs:extension>
     </xs:complexContent>
  </xs:complexType>
<xs:simpleType name="attr.contactType">
     <xs:restriction base="xs:string">
        <xs:enumeration value="technical"/>
        <xs:enumeration value="administrative"/>
        <xs:enumeration value="billing"/>
        <xs:enumeration value="other"/>
     </xs:restriction>
  </xs:simpleType>
```

2.2.4.3. `providerDescriptorType` complex type

The providerDescriptorType is a utility type, which describes generic metadata for any Liberty-enabled entity who's attributes include:

- `providerID` [required] The providerID of the entity.
- `id` [optional] The fragment identifier of the instance node (required if the node is signed (see Section 5.1).
- `validUntil` [optional] The dateTime the fragment expires. (see Section 2.2.3 [8])
- `cacheDuration` [optional] The maximum duration a consumer may cache the fragment (see Section 2.2.3 [8])
- `protocolSupportEnumeration` [required] describes the protocol release type of the entity described by providerID. NMTOKEN type allows for the enumeration of a set of liberty ID-FF protocol releases which the interfaces described within MUST support. The datatype of the tokens MUST be URN's (presently urn:liberty:iff:1.0 and urn:liberty:iff:1.1)

The elements describing the entity include:

- `ds:KeyInfo` The provider's public key.
- `SoapEndpoint` The provider's SOAP endpoint URI.
- `SingleLogoutServiceURL` The URL used for user-agent-based Single Logout Protocol profiles.
- `SingleLogoutServiceReturnURL` The URL to which the provider redirects at the end of user-agent-based Single Logout Protocol profiles.
- `FederationTerminationServiceURL` The URL used for user-agent-based Federation Termination Notification Protocol profiles.
- `FederationTerminationServiceReturnURL` The URL to which the provider redirects at the end of user-agent-based Federation Termination Notification Protocol profiles.

- `FederationTerminationNotificationProtocolProfile` The Federation Termination Notification Protocol profiles supported by the provider. Each value of the element MUST contain a valid Federation Termination Notification Protocol profile identification URI. The absence of this element SHALL mean that provider does not support any profile of the Federation Termination Notification Protocol.

- `SingleLogoutProtocolProfile` The Single Logout Protocol profiles supported by the provider. Each element MUST contain a valid Single Logout Protocol profile identification URI. The absence of this element SHALL mean that the provider does not support any profile of the Single Logout Protocol.

- `RegisterNameIdentifierProtocolProfile` The provider's preferred Register Name Identifier Protocol profile, which should be used by other providers when registering a new identifier. Each element MUST contain a valid Register Name Identifier Protocol profile identification URI. The absence of this element SHALL mean that the provider does not support any profile of the Register Name Identifier Protocol.

- `RegisterNameIdentifierServiceURL` The URL used for user-agent-based Register Name Identifier Protocol profiles.

- `RegisterNameIdentifierServiceReturnURL` The provider's redirecting URL for use after HTTP name registration has taken place.

- `RelationshipTerminationNotificationProtocolProfile` an unbounded URI type describing the profile(s) the entity supports for relationship termination as described in *ID-FF*

- `NameIdentifierMappingBinding` of type saml:AuthorityBindingType, describing the SAML authority binding at the identity provider to which identifier mapping queries can be sent.

- `Organization` The Organization (see Section 2.2.4.1) information about the provider.

- `AdditionalMetaLocation` The location of other relevant metadata about the provider.

- `libMD:ContactPerson` A Container expressing a contact person(s) responsible for technical, administrative, billing, or other information concerning an identity service implementation expressed in the metadata (see Section 2.2.4.2)

- `Extension` Provides for metadata extensions describing an *SP* or *IDP*

- `ds:Signature` An optional signature of the provider metadata (see Section 5.1)

The schema fragment for `providerDescriptorType`:

```
 <xsd:complexType name="providerDescriptorType">
      <xsd:sequence>
         <xsd:element ref="ds:KeyInfo" minOccurs="0"/>
         <xsd:element name="SoapEndpoint" type="xsd:anyURI" minOccurs="0"/>

         <xsd:element name="SingleLogoutServiceURL" type="xsd:anyURI" minOccurs="0"/>
         <xsd:element name="SingleLogoutServiceReturnURL" type="xsd:anyURI" minOccurs="0"/>
         <xsd:element name="FederationTerminationServiceURL" type="xsd:anyURI" minOccurs="0"/>
         <xsd:element name="FederationTerminationServiceReturnURL" type="xsd:anyURI" minOccurs="0"/>
         <xsd:element name="FederationTerminationNotificationProtocolProfile" type="xsd:anyURI" minOccurs="0"<
maxOccurs="unbounded"/>
         <xsd:element name="SingleLogoutProtocolProfile" type="xsd:anyURI" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="RegisterNameIdentifierProtocolProfile" type="xsd:anyURI" minOccurs="0"<
maxOccurs="unbounded"/>
         <xsd:element name="RegisterNameIdentifierServiceURL" type="xsd:anyURI" minOccurs="0"/>
         <xsd:element name="RegisterNameIdentifierServiceReturnURL" type="xsd:anyURI" minOccurs="0"/>
         <xsd:element name="RelationshipTerminationNotificationProtocolProfile" type="xsd:anyURI" minOccurs="0"<
maxOccurs="unbounded"/>
         <xsd:element name="NameIdentifierMappingBinding" type="saml:AuthorityBindingType" minOccurs="0"<
maxOccurs="unbounded"/>
         <xsd:element ref="libMD:Organization" minOccurs="0"/>
 <xsd:element name="AdditionalMetaLocation" type="xsd:anyURI" minOccurs="0" maxOccurs="unbounded"/>
```

```
        <xsd:element ref="libMD:ContactPerson" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="Extension" minOccurs="0"/>
        <xsd:element ref="ds:Signature" minOccurs="0"/>
      </xsd:sequence>

      <xsd:attribute ref="libMD:providerID" use="required"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attribute ref="libMD:validUntil" use="optional"/>
      <xsd:attribute ref="libMD:cacheDuration" use="optional"/>
 <xs:attribute name="protocolSupportEnumeration" type="NMTOKENS" use="required"/>
   </xsd:complexType>
```

2.2.4.4. `SPDescriptor` element

SPDescriptor extends `providerDescriptorType` with the following elements:

- `AssertionConsumerServiceURL` [required] One or more URI(s) of the SP for receiving Authentication Assertions from an authenticating party. When an SP sends an *AuthNRequest* to the IDP, it may indicate the preferred `AssertionConsumerServiceURL` using the provided `id` (QNAME) attribute to direct the principal to for consumption of the *AuthNResponse*.
  IDP's should inspect the Service Providers metadata for the appropriate URL, or the the default (indicated by the `isDefault` attribute) location, if no `id` is provided. Publishers MUST express only one default `AssertionConsumerServiceURL`. `AssertionConsumerServiceURL` Requires the following attributes:

  - `id` [required] the fragment identifier of the `AssertionConsumerServiceURL` used as a reference in an AuthNRequest.
  - `isDefault` [required] boolean indicator for the default `AssertionConsumerServiceURL` value to use when no identifier is provided in the request.

- `AuthnRequestsSigned` [required] boolean element indicating whether the Service Provider will always signed it's *AuthNRequest*s

the schema fragment for SPDescriptor:

```
  <xsd:element name="SPDescriptor" type="libMD:SPDescriptorType"/>
  <xsd:complexType name="SPDescriptorType">
     <xsd:complexContent>
        <xsd:extension base="libMD:providerDescriptorType">
           <xsd:sequence>
              <xsd:element name="AssertionConsumerServiceURL" type="xsd:anyURI" minOccurs="1" maxOccurs="unbounded">

                 <xsd:attribute name="id" type="xsd:ID" use="required"/>
                 <xsd:attribute name="isDefault" type="xsd:boolean" use="required" default="false"/>
              </xsd:element>

              <xsd:element name="AuthnRequestsSigned" type="xsd:boolean"/>

           </xsd:sequence>
        </xsd:extension>
     </xsd:complexContent>
  </xsd:complexType>
```

2.2.4.4.1. SPDescriptor Example

[Need example]

2.2.4.5. `IDPDescriptor` element

`IDPDescriptor` extends `providerDescriptorType` with the following elements:

- `SingleSignOnServiceURL` [required] The identity provider s URL for accepting authentication requests for the Single Sign-On and Federation Protocol.
- `SingleSignOnProtocolProfile` [required] The Single Sign-On Protocol profiles supported by the provider. Each element MUST contain a valid Single Sign-On Protocol profile identification URI.
- `IntroductionNotificationProtocolProfile` of URI type describes the profile of this protocol supported by the identity provider.

The schema fragment for `IDPDescriptor`:

```
<xsd:element name="IDPDescriptor" type="libMD:IDPDescriptorType"/>
<xsd:complexType name="IDPDescriptorType">
   <xsd:complexContent>
      <xsd:extension base="libMD:ProviderDescriptorType">
         <xsd:sequence>
            <xsd:element name="SingleSignOnServiceURL" type="xsd:anyURI"/>
            <xsd:element name="SingleSignOnProtocolProfile" type="xsd:anyURI" maxOccurs="unbounded"/>
<xsd:element name="IntroductionNotificationProtocolProfile" type="xsd:anyURI" minOccurs="0" maxOccurs="unbounded"/>

         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

2.2.4.5.1. IDPDescriptor Example

[Need Example]

2.2.4.6. `ServiceInstanceDescriptor` element

The `ServiceInstanceDescriptor` element element conveys metadata concerning a one or more *ID-WSF* service instance. It extends *isf:ServiceInstanceDescriptorType* with the following attributes:

- `providerID` [required] the providerID operating the service instance.
- `id` [optional] fragment identifier (required if the node is signed)
- `validUntil` [optional] the expiration date of the metadata node.
- `cacheDuration` [optional] the maximum cache dateTime for the metadata node.

and the following elements:

- `ContactPerson` [optional] one or more contacts responsible for the service instance.
- `AdditionalMetaLocation` [optional] A URL where additional metadata may be obtained for the service instance.
- `Extension` optional extension point for conveying additional metadata concerning the service instance
- `ds:Signature` [optional] Signature of the `ServiceInstanceDescriptor` node.

The schema fragment for `ServiceInstanceDescriptor` element

```
   <xsd:element name="ServiceInstanceDescriptor" type="libMD:serviceInstanceDescriptorType" minOccurs="0"<
maxOccurs="unbounded"/>
   <xsd:complexType name="serviceInstanceDescriptorType">
      <xsd:complexContent>
         <xsd:extension base="isf:ServiceInstanceDescriptorType">
            <xsd:sequence>
               <xsd:element ref="libMD:ContactPerson" minOccurs="0" maxOccurs="unbounded"/>
               <xsd:element name="AdditionalMetaLocation" type="xsd:anyURI" minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="Extension" minOccurs="0"/>
               <xsd:element ref="ds:Signature" minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute ref="libMD:providerID" use="required"/>
            <xsd:attribute name="id" type="xsd:ID" use="optional"/>
            <xsd:attribute ref="libMD:validUntil" use="optional"/>
            <xsd:attribute ref="libMD:cacheDuration" use="optional"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
```

2.2.4.6.1. ServiceInstanceDescriptor Example

[Need example]

2.2.4.7. `EntityDescriptor` element

The element `EntityDescriptor` is used to contain one or more descriptor types for a given organization. Publishers MUST NOT convey metadata for other unaffiliated organizations here. They should use EntitiesDescriptor (Section 2.2.4.8) instead. Publishers MUST publish all relevant roles in this single document, or indirectly through `AdditionalMetaLocation`.

Note that it is possible for a single organization to be represented by more than one `providerID`, by indicating different providerID attributes for each providerDescriptor.

Attributes for `EntityDescriptor`:

- `id` [optional] fragment identifier required if `ds:Signature` is present.
- `validUntil` The expiration dateTime of the metadata.
- `cacheDuration` The cache duration period for the metadata.

Elements contained in `EntityDescriptor`:

- `SPDescriptor` Metadata concerning a Service Provider.
- `IDPDescriptor` Metadata concerning an Identity Provider.
- `ServiceInstanceDescriptor` Metadata describing an *ID-WSF* service endpoint.
- `AffiliationDescriptor` Metadata concerning an affiliation entity
- `Extension` provides extension point for additional entity metadata
- `ContactPerson` Contact information for the overall entity (see Section 2.2.4.2.
- `Organization` Organizational information about the entity (see Section 2.2.4.1).

- `ds:Signature` An XML Signature on the entire entity metadata instance.

The schema fragment for `EntityDescriptorType`:

```
  <xsd:element name="EntityDescriptor" type="libMD:entityDescriptorType"/>
  <xsd:complexType name="entityDescriptorType" mixed="false">
     <xsd:sequence maxOccurs="unbounded">

        <xsd:element ref="libMD:SPDescriptor" minOccurs="0"/>
        <xsd:element ref="libMD:IDPDescriptor" minOccurs="0"/>
        <xsd:element ref="libMD:ServiceInstanceDescriptor" minOccurs="0"/>
<xsd:element ref="libMD:AffiliationDescriptor" minOccurs="0"/>
<xsd:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="libMD:ContactPerson" minOccurs="0"/>
        <xsd:element ref="libMD:Organization" minOccurs="0"/>
        <xsd:element ref="ds:Signature" minOccurs="0" />
     </xsd:sequence>
     <xsd:attribute name="id" type="xsd:ID" use="optional"/>
     <xsd:attribute ref="libMD:validUntil" use="optional"/>
     <xsd:attribute ref="libMD:cacheDuration" use="optional"/>
  </xsd:complexType>
```

2.2.4.8. `EntitiesDescriptor`

The element `EntitiesDescriptor` describes more than one organization in a single instance document. It is simply described as 2 or more `EntityDescriptors`.

The schema fragment for `EntitiesDescriptor` element:

```
  <xsd:element name="EntitiesDescriptor" type="entitiesDescriptorType"/>
  <xsd:complexType name="entitiesDescriptorType">
     <xsd:sequence>
        <xsd:element ref="libMD:EntityDescriptor" minoccurs="2" maxoccurs="unbounded"/>
     </xsd:sequence>
  </xsd:complexType>
```

2.2.4.9. `AffiliationDescriptor`

The `AffiliationDescriptor` element describes a group of entities, identified collectively by `affiliationID`, as an enumeration of `providerID`'s. The same uniqueness constraints for `affiliationID` apply, that is, it MUST be unique across all Liberty entities with which the affiliation expects to interact, therefore, it MUST NOT be the `providerID` of any of the members of the affiliation. It is the responsibility of the entity represented by `affiliationOwnerID` to administer this identifier.

`AffiliationDescriptor` element contains the following attributes:

- `affiliationID` [required] the identifier for the affiliation (with identical structure constraints as `providerID`. See Section 2.2.2)
- `affiliationOwnerID` [required] the `providerID` of the owner or parent operator of the the affiliation, from which, additional metadata may be derived

and the following elements:

- `AffiliateMember` [required] One or more providers who are members of the affiliation. The value should be a `providerID` who's metadata may be obtained via methods described in Section 3

- `Extension` provides an extension point to convey additional metadata concerning the affiliation
- `ds:KeyInfo` [optional] Zero or one public key material reference that is the property of the affiliation. This keying material SHOULD be separate from the keying material of the `providerID` who may be referenced as the `affiliateOwnerID`
- `ds:Signature` [optional] An XML Signature of the metadata node `AffiliationDescriptor`.

The schema fragment for the `AffiliationDescriptor` element:

```
 <xsd:element name="AffiliationDescriptor" type="affiliationDescriptorType"/>
 <xsd:complexType name="affiliationDescriptorType">
 <xsd:sequence>
 <xsd:element name="AffiliateMember" type="libMD:entityIDType" minOccurs="1" maxOccurs="unbounded"/>
 <xsd:element ref="Extension" minOccurs="0"/>
 <xsd:element ref="ds:KeyInfo" minOccurs="0"/>
 <xsd:element ref="ds:Signature" minOccurs="0" maxOccurs="1"/>
 </xsd:sequence>
 <xsd:attribute name="affiliationID" type="libMD:entityIDType" use="required"/>
 <xsd:attribute name="affiliationOwnerID" type="libMD:entityIDType" use="required"/>
 </xsd:complexType>
```

2.3. Complete Metadata Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:metadata:1.2"
   xmlns="urn:liberty:metadata:1.2"
   xmlns:isf="urn:liberty:wsf:disco:1.0"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
   xmlns:pip="urn:liberty:idpp:1.0"
   elementFormDefault="qualified" attributeFormDefault="unqualified">
   <!-
  $Id: draft-arch-metadata-schema.xsd,v 1.8 2003/04/14 04:50:45 pdavis Exp $
  Author: Peter Davis
  Last editor: $Author: pdavis $
  $Date: 2003/04/14 04:50:45 $
  $Revision: 1.8 $
  ->
   <xs:import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="http://www.w3.org/TR/xmldsig-
core/xmldsig-core-schema.xsd"/>
   <xs:import namespace="urn:liberty:wsf:disco:1.0" schemaLocation="lib-arch-disco-svc.xsd"/>

   <xs:import namespace="urn:liberty:idpp:1.0" schemaLocation="lib-svc-id-pp.xsd"/>
   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
      schemaLocation="http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-assertion-01.xsd"/>

   <xs:include schemaLocation="lib-arch-iwsf-utility.xsd"/>

   <xs:simpleType name="entityIDType">
      <xs:restriction base="xs:anyURI">
         <xs:maxLength id="maxlengid" value="1024"/>
         <xs:pattern value=""/>
      </xs:restriction>
   </xs:simpleType>

   <xs:attribute name="libertyPrincipalIdentifier" type="entityIDType"/>
   <xs:attribute name="providerID" type="entityIDType"/>
   <xs:attribute name="validUntil" type="xs:dateTime"/>
   <xs:attribute name="cacheDuration" type="xs:duration"/>
   <xs:element name="Organization" type="organizationType"/>
   <xs:complexType name="organizationType">
      <xs:sequence>
         <xs:element name="OrganizationName" type="xs:string"/>
         <xs:element name="OrganizationDisplayName" type="xs:string"/>
         <xs:element name="OrganizationURL" type="xs:anyURI"/>
 <xs:element ref="Extension" minOccurs="0"/>
      </xs:sequence>
```

```
    </xs:complexType>
    <xs:element name="ContactPerson" type="contactType"/>
    <xs:complexType name="contactType">
       <xs:complexContent>
          <xs:extension base="pip:IDPPType">
             <xs:attribute ref="libertyPrincipalIdentifier" use="optional"/>
             <xs:attribute name="contactType" default="technical" use="required" type="attr.contactType"/>
          </xs:extension>
       </xs:complexContent>
    </xs:complexType>
 <xs:simpleType name="attr.contactType">
       <xs:restriction base="xs:string">
          <xs:enumeration value="technical"/>
          <xs:enumeration value="administrative"/>
          <xs:enumeration value="billing"/>
          <xs:enumeration value="other"/>
       </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="providerDescriptorType">
       <xs:sequence>
          <xs:element ref="ds:KeyInfo" minOccurs="0"/>
          <xs:element name="SoapEndpoint" type="xs:anyURI" minOccurs="0"/>
          <!- ISSUE: establishing SoalEndPoint with ?concrete? wsdl ->
          <xs:element name="SingleLogoutServiceURL" type="xs:anyURI" minOccurs="0"/>
          <xs:element name="SingleLogoutServiceReturnURL" type="xs:anyURI" minOccurs="0"/>
          <xs:element name="FederationTerminationServiceURL" type="xs:anyURI" minOccurs="0"/>
          <xs:element name="FederationTerminationServiceReturnURL" type="xs:anyURI" minOccurs="0"/>
          <xs:element name="FederationTerminationNotificationProtocolProfile" type="xs:anyURI" minOccurs="0"<
maxOccurs="unbounded"/>
          <xs:element name="SingleLogoutProtocolProfile" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="RegisterNameIdentifierProtocolProfile" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>

          <xs:element name="RegisterNameIdentifierServiceURL" type="xs:anyURI" minOccurs="0"/>
          <xs:element name="RegisterNameIdentifierServiceReturnURL" type="xs:anyURI" minOccurs="0"/>
          <xs:element name="RelationshipTerminationNotificationProtocolProfile" type="xs:anyURI" minOccurs="0"<
maxOccurs="unbounded"/>
          <xs:element name="NameIdentifierMappingBinding" type="saml:AuthorityBindingType" minOccurs="0"<
maxOccurs="unbounded"/>
          <xs:element ref="Organization" minOccurs="0"/>
          <xs:element name="AdditionalMetaLocation" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="ContactPerson" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="Extension" minOccurs="0"/>
          <xs:element ref="ds:Signature" minOccurs="0"/>
       </xs:sequence>
       <xs:attribute ref="providerID" use="required"/>
       <xs:attribute name="id" type="xs:ID" use="optional"/>
       <xs:attribute ref="validUntil" use="optional"/>
       <xs:attribute ref="cacheDuration" use="optional"/>
 <xs:attribute name="protocolSupportEnumeration" type="xs:NMTOKENS" use="required"/>
    </xs:complexType>
    <xs:element name="EntityDescriptor" type="entityDescriptorType"/>
    <xs:complexType name="entityDescriptorType" mixed="false">
       <xs:sequence maxOccurs="unbounded">
          <xs:element ref="SPDescriptor" minOccurs="0"/>
          <xs:element ref="IDPDescriptor" minOccurs="0"/>
          <xs:element ref="ServiceInstanceDescriptor" minOccurs="0"/>
          <xs:element ref="AffiliationDescriptor" minOccurs="0"/>
          <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="ContactPerson" minOccurs="0"/>
          <xs:element ref="Organization" minOccurs="0"/>
          <xs:element ref="ds:Signature" minOccurs="0"/>
       </xs:sequence>
       <xs:attribute name="id" type="xs:ID" use="optional"/>
       <xs:attribute ref="validUntil" use="optional"/>
       <xs:attribute ref="cacheDuration" use="optional"/>
    </xs:complexType>
    <xs:element name="SPDescriptor" type="SPDescriptorType"/>
    <xs:complexType name="SPDescriptorType">
       <xs:complexContent>
          <xs:extension base="entityDescriptorType">
             <xs:sequence>
 <xs:element name="AssertionConsumerServiceURL" minOccurs="1" maxOccurs="unbounded">
 <xs:complexType>
 <xs:simpleContent>
 <xs:extension base="xs:anyURI">
 <xs:attribute name="id" type="xs:ID" use="required"/>
 <xs:attribute name="isDefault" type="xs:boolean" use="required" default="false"/>
```

```
 </xs:extension>
 </xs:simpleContent>
 </xs:complexType>
 </xs:element>
 <xs:element name="AuthnRequestsSigned" type="xs:boolean"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="IDPDescriptor" type="IDPDescriptorType"/>
   <xs:complexType name="IDPDescriptorType">
      <xs:complexContent>
         <xs:extension base="entityDescriptorType">
            <xs:sequence>
               <xs:element name="SingleSignOnServiceURL" type="xs:anyURI"/>
               <xs:element name="SingleSignOnProtocolProfile" type="xs:anyURI" maxOccurs="unbounded"/>
               <xs:element name="IntroductionNotificationProtocolProfile" type="xs:anyURI" minOccurs="0"<
maxOccurs="unbounded"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="ServiceInstanceDescriptor" type="serviceInstanceDescriptorType" minOccurs="0"<
maxOccurs="unbounded"/>
   <xs:complexType name="serviceInstanceDescriptorType">
      <xs:complexContent>
         <xs:extension base="isf:ServiceInstanceType">
            <xs:sequence>
               <xs:element ref="ContactPerson" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element name="AdditionalMetaLocation" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="Extension" minOccurs="0"/>
               <xs:element ref="ds:Signature" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute ref="providerID" use="required"/>
            <xs:attribute name="id" type="xs:ID" use="optional"/>
            <xs:attribute ref="validUntil" use="optional"/>
            <xs:attribute ref="cacheDuration" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="EntitiesDescriptor" type="entitiesDescriptorType"/>
   <xs:complexType name="entitiesDescriptorType">
      <xs:sequence>
         <xs:element ref="EntityDescriptor" minoccurs="2" maxoccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="AffiliationDescriptor" type="affiliationDescriptorType"/>
   <xs:complexType name="affiliationDescriptorType">
      <xs:sequence>
         <xs:element name="AffiliateMember" type="entityIDType" minOccurs="1" maxOccurs="unbounded"/>
         <xs:element ref="Extension" minOccurs="0"/>
         <xs:element ref="ds:KeyInfo" minOccurs="0"/>
         <xs:element ref="ds:Signature" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="affiliationID" type="entityIDType" use="required"/>
      <xs:attribute name="affiliationOwnerID" type="entityIDType" use="required"/>
 <xs:attribute name="id" type="xs:ID" use="optional"/>
   </xs:complexType>

</xs:schema>
```

3. Publishing the Metadata

Two mechanisms are provided for entities to publish metadata document locations: via the DNS and via a "well-known-location", via direct dereferencing of the `providerID` itself.

In either case, when retrieval requires network transport of the document, the transport SHOULD be protected with *TLS/SSL* in order to ensure integrity of the metadata document, as among other information within the document, trust establishment may be based in part on information provided here. Relying parties of this metadata SHOULD process the *TLS/SSL* Certificate presented by the server through normal validation procedures.

Trust establishment of the MetaData will be based on one or more of: DNS signatures (RECOMMENDED), TLS server authentication (RECOMMENDED), and MetaData `ds:Signature` (STRONGLY RECOMMENDED) evaluations. Publishers MAY implement additional trust mechanism, in conjunction with the required suggested server authentication. Additional trust metadata content, if supplied, MUST be placed in the extension points provided.

3.1. Using the DNS to publish metadata location(s)

In order to ensure that all providers have accessible metadata locations, entities are STRONGLY RECOMMENDED to publish their metadata document locations in a zone of their corresponding DNS. As *providerID*s are flexible identifiers, publication and resolution is determined by an entities URI scheme and fully qualified name part of the identifier.

URI locations for metadata will then be derived through queries of the NAPTR Resource Record (RR) as defined in *RFC2915* and *RFC3403*.

It is SUGGESTED that entities publish their resource records in signed zone files using *DNSSEC* such that relying parties may establish certain trust decisions based on these signatures, specifically, the validity of the resource record itself. If DNS zone signatures are present, relying parties MUST properly validate the signature.

3.1.1. Publication of Metadata locations

Readers are encouraged to read *RFC2915* and *RFC3403* to gain familiarity with this resource record, as this specification is a direct profile of them.

DDDS (*RFC3401* through *RFC3405*) is a general purpose system for the retrieval of information based on an application specific input string and applying well known rules to transform that string until a terminal condition is reached requiring a look up into a application specific defined database or execution of a URL based on the application defined rules. DDDS defines a specific type of definable DNS Resource Record, NAPTR records, for the storage in the DNS of information necessary to apply DDDS rules.

Entities MAY publish separate URL's when the metadata documents need to be distributed, or where different metadata documents are required due to multiple Authentication Domain memberships which require separate keying material, or where service interfaces require separate metadata declarations. This may be accomplished through the use of the optional `AdditionalMetaLocation` attribute in the core or other subordinate metadata document, or through the regexp facility and multiple service definition fields in the NAPTR resource record itself.

If `providerID` is a URN, resolution of the MetadataLocation proceeds as specified in *RFC3404*. Otherwise, the resolution of the metadata location proceeds as specified in this specification.

Following is the application specific descriptions for the DDDS application for the Liberty Metadata resolution protocols.

3.1.1.1. Application Unique String

Liberty metadata resolution shall begin with the application unique string of `providerID`

3.1.1.2. First Well Known Rule

The "first well-known-rule" for processing Liberty Alliance Metadata resolution is to parse the `providerID` URI and extract the fully qualified domain name (subexpression 3) as described in section Section 4.1.1

3.1.1.3. The Order field

The order field indicates the order for processing each NAPTR resource record returned. Publishers MAY provide multiple NAPTR resource record's which MUST be processed by the resolver application in the order indicated by this field.

3.1.1.4. The Preference Field

For terminal NAPTR resource record's, the publisher expresses the preferred order of use to the resolving application. The resolving application MAY ignore this order, in cases where the service field value does not meet the resolvers requirements (eg: the resource record returns a protocol the application does not support).

3.1.1.5. The Flag Field

Liberty Metadata resolution makes use of two of the "U" flag, which is terminal, and the null value (implying additional resource record's are to be processed). The "U" flag indicates that the output of the rule is a URI.

3.1.1.6. The Service Field

The Liberty specific service fields shall include:

```
servicefield = 1("PID2U" / "NID2U") "+" proto [*(":" class) *(":" servicetype)]
proto = 1("https" / "uddi")
class = 1( "entity" / "entitygroup" )
servicetype = 1(si / "sp" / "idp" / "authn" / alphanum )
si = "si" [":" alphanum] [":endpoint"]
alphanum = 1*32(ALPHA / DIGIT)
```

where

- PID2U resolves a providerID identifier to metadata URL
- NID2U resolves a nameIdentifier (principal) metadata URL
- proto describes the retrieval protocol (https or uddi). In the case of UDDI, the resulting URI will be a http(s) URI referencing a `WSDL` document.
- class identifies which indicates whether the referenced metadata document describes a single provider, or multiple. In the latter case, the referenced document MUST contain the entity defined by `providerID` as a member of a group of entities within the document itself.
- servicetype allows a publishers to publish service provider, identity provider and service instance metadata locations as separate documents. Resolvers who encounter multiple servicetype declarations will dereference the appropriate URI, depending on which service type required for an operation (eg: a provider operating both and IDP and an SP service, may publish SP and IDP metadata at different locations).
- the `si` component (with optional endpoint component) allows the publisher to directly publish either the metadata for a service instance as defined by ID-WSF 1.0 [ISF[], or articulating the soap endpoint (using `endpoint`

For example:

- PID2U+https:entity - represents the complete entity metadata document via the https protocol
- PID2U+https:entity:si:pip - returns the PIP metadata URL for the entity described by providerID via the https protocol profile
- PID2U+uddi:entity:si:foo - returns the WSDL document location which describes a service instance "foo"
- PID2U+https:entitygroup:idp - returns the metadata for a group of entities, of which providerID is a member. the referenced document describes (one or more) IdPs in the group
- NID2U+https:idp - returns an IDP providerIDs, who can provider authentication services for a principal
- NID2U+https:authn - returns a URL to attempt to authenticate the principal against

3.1.1.7. The regex and replacement fields

The expected output after processing the application unique sting through the regex MUST be a valid https URL or UDDI node (http references wsdl document) address.

3.1.2. NAPTR Examples

3.1.2.1. Provider Metadata NAPTR Examples

Entities publish metadata URLs in the following manner:

```
$ORIGIN provider.biz
;; order pref f service regexp or replacement
IN NAPTR 100 10 "U" PID2U+https:entity  "!^.*$!https://host.provider.biz/some/directory/trust.xml!" ""
IN NAPTR 110 10 "U" PID2U+https:entity:trust "!^.*!https://foo.provider.biz:1443/mdtrust.xml!" ""
IN NAPTR 125 10 "U" PID2U+https:"
IN NAPTR 110 10 "U" PID2U+uddi:entity "!^.*$!https://this.uddi.node.provider.biz/libmd.wsdl" ""
```

3.1.2.2. Name identifier examples

Principals employer example.int operates an IDP which may be used by a office supply company to authenticate authorized buyers. The supplier takes users email address buyer@example.int as input to the resolution process, and parses the email address to extract the FQDN (example.int). The employer publishes the following NAPTR in example.int:

```
$ORIGIN example.int.
IN NAPTR 100 10 "U" NID2U+https:authn "!^([^@]+)@(.*)$!https://serv.example.int:8000/cgi-bin/getmd?\1!" ""
IN NAPTR 100 10 "U" NID2U+https:idp  "!^([^@]+)@(.*)$!https://auth.example.int/app/auth?\1" ""
```

3.2. Publication via Well-Known Location

Entities MAY publish their metadata documents at a well known location. The core metadata document location in this profile simply involves directly dereferencing the providerID and obtaining the document directly (or through schema-specific means of indirection)

For well known location documents, the XML document MUST describe the metadata for the **providerID** entity only. If other entities need to be described, the **AdditionalMetaLocation** MUST be used. Thus the **entitiesDescriptor** MUST NOT be used in documents published at a well know location, since entities as a group, are not defined by such an identifier.

4. Metadata Resolution and Retrieval

Metadata publication is provided for in two fashions: via a "well-known-location" and via queries on the DNS. Both mechanisms depend upon the processing of the `providerID` element (see [Section 3]), which is the primary identifier for Liberty-enabled entities.

The providerID, is defined as a restricted form of `anyURI` Section 2.2.2, therefore, shall be parsed as in Section 4.1.1 for these resolution profiles.

4.1. Resolving Locations and Retrieving Metadata

The summarized steps for retrieving metadata from a given `providerID` is as follows:

- (optionally) attempt locating the metadata document(s) via the *well known location* profile by directly dereferencing the providerID (end if a document was located, validated and fulfills metadata requirements for present operations)
- If the providerID is a URN, proceed resolution steps as defined in *RFC3404*
- parse the `providerID` to obtain the FQDN
- query the DNS for NAPTR resource record's of the `domain name` iteratively until a terminal resource record is returned
- identify which resource record to use based on the service fields, then order fields, then preference fields of the result set
- obtain the document(s) at the provided location(s) as required by the application

4.1.1. Parsing the ProviderID

To initiate the resolution of the location of the target metadata elements, it will be necessary in some cases to decompose the ProviderID (expressed as a URI) into one or more atomic elements.

The following regular expression should be used when initiating the decomposition process:

```
^([^:/?#]+:)?/*([^:/?#]*@)?(([^/?:#]*\.)*(([^/?#:\.]+)\.([^/?#:\.]+)))(:\d+)?([^?#]*)(\?[^#]*)?(#.*)?$
 1         2        34        56         7         8    9     10      11
```

Subexpression 3 MUST result in a Fully Qualified Domain Name (FQDN), which will be the basis for retrieving metadata locations from this zone.

4.1.2. Obtaining metadata via the DNS

Upon completion of the parsing of the `providerID`, the application then performs a DNS query for resulting domain (subexpression 5), for NAPTR resource record's, for which it should expect 1 or more responses. Applications MAY exclude from the result set any service definitions which do not concern the present request operations.

Resolving applications MUST then order the result set according to the order field, and MAY order the result set based on the preference set. Resolvers are NOT REQUIRED to follow the ordering of the preferences field.

The resulting NAPTR resource record(s) are operated on iteratively (based on the order flag), until a terminal NAPTR resource record is reached.

The result will be a well formed, fully qualified URL, which will then be used to retrieve the metadata document.

4.1.2.1. Post Processing Operations

When service specific metadata is sought, resolvers MAY filter the NAPTR result set based on more specific resource record's with service identifiers which match the service(s) sought.

4.1.3. Obtaining Metadata via the "Well-Known Location method"

Consumers of published metadata MAY attempt retrieval via the well-known-location method by directly dereferencing the providerID.

Other forms of well-known location MAY be agreed upon by a group of Liberty entities, however, it is STRONGLY SUGGESTED that publication in the DNS be employed as well, to allow for interactions with other Liberty *ID-WSF 1.0* implementations.

The resulting XML document MUST describe the metadata for the **providerID** entity only. If other entities need to be described, the **AdditionalMetaLocation** MUST be used.

There may be only one location, although this document MAY point to other document locations using the `AdditionalMetaLocation` element.

5. Post Processing of the Metadata document

5.1. Processing of `ds:Signature` and general trust processing

Metadata processing provides several mechanisms for trust negotiation for both the metadata itself and the trust ascribed to the entity described by such metadata:

- Trust derived from the signature of the zone from which the metadata location URI was resolved, ensuring accuracy of the metadata document location(s)
- Trust derived from signature processing of the metadata document itself, ensuring the integrity of the XML document, especially in cases where it may be locally cached
- Trust derived from the SSL/TLS negotiation of the metadata delivery URI, ensuring the identity of the publisher of the metadata

Post processing of the metadata document SHOULD include at least one of these processes. Specifically, the relying party MAY choose to trust any of the cited authorities in the resolution and parsing process. Publishers of metadata MAY employ any of these processing profiles to establish trust of the subject of the metadata document, governed by implementation policies

5.1.1. Processing signed DNS zones

Verification of zone signature SHOULD be processed, if present, as described in *DNSSEC*

5.1.2. Processing signed documents and fragments

Published metadata documents SHOULD be signed, either by a certificate issued to the subject of the document, or by another trusted party. Publishers can consider signatures of other parties as an alternative to trust conveyance.

Consumers MUST validate signatures of the metadata document on initial retrieval as well as each time it is retrieved from a local cache as described by *XMLDSIG*, in order to avoid local document tampering.

5.1.3. Processing Server Authentication in MetaData Retrieval via TLS/SSL

It is STRONGLY RECOMMENDED that publishers implement TLS URL's, therefore consumers SHOULD consider the trust inherited from the issuer of the TLS/SSL certificate. Since publication URLs may not always located in the domain of the provider of the subject of the metadata document, consumers SHOULD NOT expect certificates whose subject is the provider, as it may be hosted at another trusted party.

Also, since the basis of this trust may not be available against a cached document, other mechanisms SHOULD be used under such circumstances.

5.2. Metadata Location and Document Caching

Location caching based on DNS profiles MUST NOT exceed the TTL of the DNS zone from which the location was derived. Resolvers MUST obtain a fresh copy of the MetaData location upon reaching the expiration of the TTL of the zone.

Publishers of Metadata documents should carefully consider the TTL of the zone when making updates to it's metadata document location. Should such a location change occur, publishers MUST either keep the document at both the old and new location until all conforming resolvers are certain to have the updated location (eg: time of zone change + TTL), or provide an `HTTP Redirect` to the new location.

Document caching MUST NOT exceed the `validUntil` attribute of the subject element(s) and the `cacheDuration` attribute. If fragments have parents which contain caching policies, the parent fragment ALWAYS takes precedence.

Consumers MUST retain the *dateTime* when the document was retrieved, in order to properly process the `cacheDuration` attributes on fragments and documents.

When a document or fragment has expired, the consumer MUST retrieve a fresh copy, which may require a refresh of the document location(s). Consumers SHOULD process document cache processing according to *RFC2616* section 13 (HTTP 1.1), and MAY request the Last-Modified dateTime from the HTTPS server. Publishers SHOULD ensure acceptable cache processing as described in *RFC2616* (Section 10.3.5 304 Not Modified)

5.3. Handling of HTTPS Redirects

Publishers MAY issue an HTTP Redirect (301 Moved Permanently, or 307 Temporary Redirect), and user agents MUST follow the specified URL in the Redirect response.

Redirects SHOULD be to a TLS/SSL protected resource, and SHOULD be of the same protocol as the initial request.

6. Security Considerations

6.1. Trust Establishment

Cryptographic signatures are used to establish identity and tamper evidence in several locations within the metadata specification. While valid signatures convey some level of trust in the resulting document, extreme care should be taken as to the validity of the URIs described within the document itself. Relying parties should carefully inspect agreements and statements made by the signing authorities of the subject certificates or keys.

7. References