CIQ TC Specifications

Customer Information Quality Technical Committee

Name, Address and Party Information Technical Specification

Version 3.0 (draft)

Date Created: 15 August 2004

Last Updated: 30 November 2005

Editors

Max Voskob, Individual Member, OASIS CIQ TC (max.voskob@paradise.net.nz)

Ram Kumar, Individual Member and Chair, OASIS CIQ TC (kumar.sydney@gmail.com)

Contributors

John Glaubitz	Vertex	Member, CIQ TC
Hido Hasimbegovic	Individual	Member, CIQT TC
Robert James	Individual	Member, CIQ TC
Joe Lubenow	Individual	Member, CIQ TC
Mark Meadows	Microsoft Corporation	Member, CIQ TC
John Putman	Individual	Prospective Member, CIQ TC
Michael Roytman	Vertex	Member, CIQ TC
Colin Wallis	New Zealand Government	Member, CIQ TC
David Webber	Individual	Member, CIQ TC

Abstract

This Technical Specification defines the name (xNL), address (xAL), name and address (xNAL) and Party Information (xPIL) specifications version 3.0.

Intellectual Property Rights, Patents, Licenses and Royalties

CIQ TC Specifications (includes documents, schemas and examples[1 and 2]) are free of any Intellectual Property Rights, Patents, Licenses or Royalties. Public is free to download and implement the specifications free of charge. Please, read OASIS Copyright Notice in APPENDIX A.

[1]xAL-Australia.XML

Address examples come from AS/NZ 4819:2003 standard of Standards Australia and are subject to copyright

[2]xAL-international.xml

Address examples come from a variety of sources including Universal Postal Union (UPU) website and the UPU address examples are subject to copyright.

TABLE OF CONTENTS

1 Schema design approach in version 3.0

Name, Address and Party schemas of version 3.0 share the same design concepts. The commonality should simplify understanding and adoption of the schemas. xNAL schema stands out as it is only a simple container for associating names and addresses.

Name, Address and Party schemas were designed to bring interoperability the way these most “common” entities are used across all spectrums of business and government.

1.1 Version 3.0 schema files

Following are the different schemas produced for version 3.0:

Schema File name	Description	Comments
xNL.xsd	Entity Name	Defines a set of reusable types and elements for a name of individual or organisation
xNL-types.xsd	Entity Name	Defines a set of enumerations that suit this particular application
xAL.xsd	Entity Address	Defines a set of reusable types and elements for an address, location name or description
xAL-types.xsd	Entity Address	Defines a set of enumerations that suit this particular application
xNAL.xsd	Name and Address binding	Defines two constructs to bind names and addresses for data exchange or postal purposes
xPIL.xsd (formerly xCIL.xsd)	Entity Party (organisation or individual)	Defines a set of reusable types and elements for a detailed description of an organisation or individual
xPL-types.xsd	Entity Party (organisation or individual)	Defines a set of enumerations that suit this particular application
xLink.xsd	xLink attributes	Defines a subset of xLink attributes as XML schema
xPRL.xsd (formerly xCRL.xsd)	Party relationships	Defines a simple reusable type for party relationships (not currently utilised)

1.2 Formal design requirements for version 3.0

Following are the formal design requirements taken into consideration for version 3.0 schemas:

- Data structures should be described using W3C XML Schema language
- Data structures should be separated into multiple namespaces for reuse of the main fundamental entities (e.g. Person Name, Organisation Name, Address)
- Data structures should be able to accommodate all information types used for data exchanges based on previous versions of the CIQ Specifications
- Data structures should be extensible (also, allow reduction in complexity) to provide enough flexibility for point-to-point solutions and application-specific scenarios

- Data structures should allow organisation-specific information to be attached to entities without breaking the structure
- Implementation complexity should be proportional to the complexity of the subset of data structures used by the implementer

1.3 Major entities

The entire party information space is divided into a number of complex information types that are viewed as basic entities. This enables re-use of the basic entities as required. Following are the entities:

- Name (see xNL.xsd, xNL-types.xsd)
- Address (see xAL.xsd, xAL-types.xsd)
- Name and Address combined (see xNAL.xsd)
- Personal details and specifics (see xPIL.xsd, xPIL-types.xsd)
- Organisation details and specifics (see xPIL.xsd, xPIL-types.xsd)
- Party Relationships (see xPRL.xsd and xLink.xsd)

1.4 Common approaches

The design concepts of name, address and party schemas are very similar in terms of the way semantic information (e.g. Semantic information for a person name is “Given Name, “Middle Name’ Surname” etc, i.e. adding semantics to the data) is represented. All the common concepts are explained in section 2 (Entity “Name”). It is recommended to study that section in detail before proceeding to other entities.

1.5 Namespaces

Entity	Namespace	Recommended prefix	Schema files
Name	urn:oasis:names:tc:ciq:xnl:3	xnl or n	xNL.xsd xNL-types.xsd
Address	urn:oasis:names:tc:ciq:xal:3	xal or a	xAL.xsd xAL-types.xsd
Name and address	urn:oasis:names:tc:ciq:xnal:3	xnal	xNAL.xsd
Party	urn:oasis:names:tc:ciq:xpil:3	xpil or p	xPIL.xsd xPIL-types.xsd
Party relationships	urn:oasis:names:tc:ciq:xprl:3	xprl or r	xPRL.xsd
xLink	http://www.w3.org/1999/xlink	xlink	xLink.xsd

1.6 Other specifications

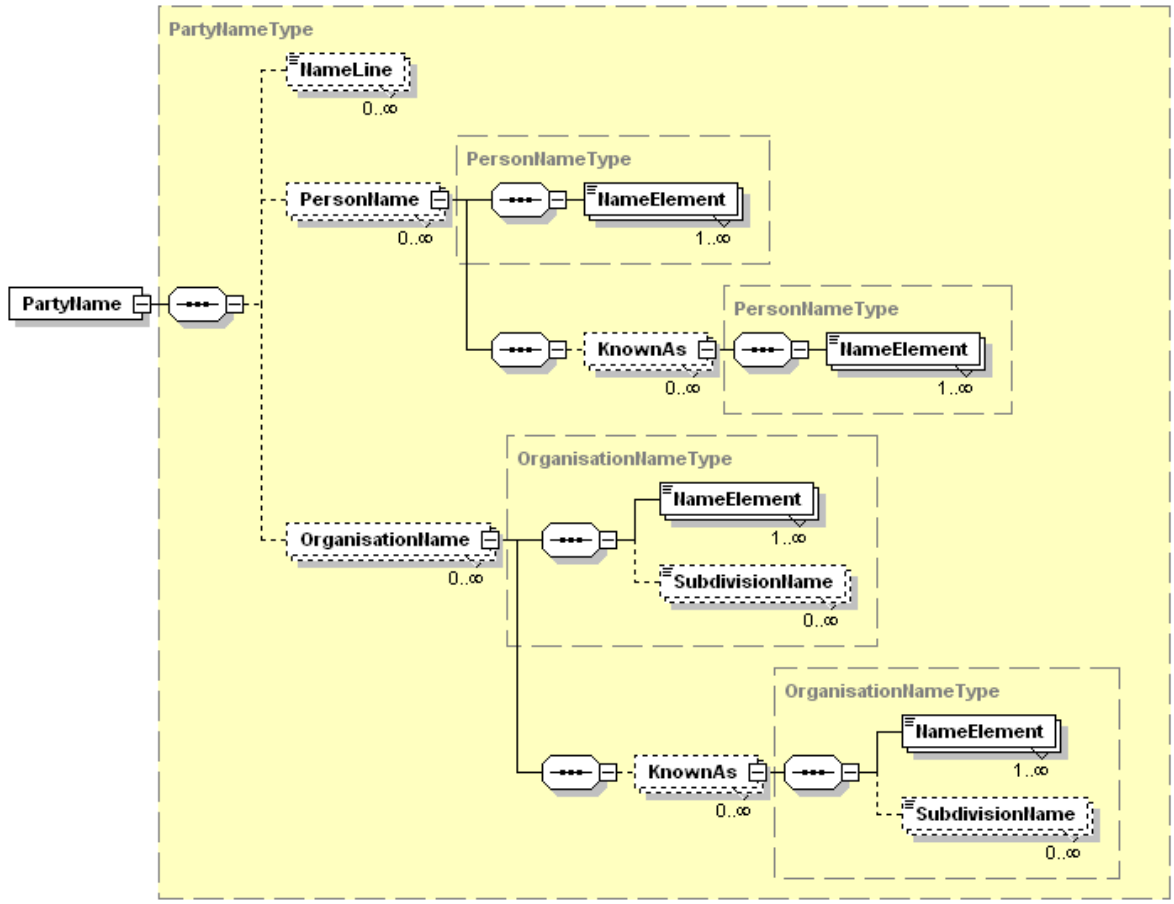
This document contains references to XML Linking Language (XLink) Version 1.0, W3C Recommendation 27 June 2001 available at http://www.w3.org/TR/xlink/

2 Entity “Name”

Entity “*Name*” has been modelled independent of any context as a standalone class to reflect some common understanding of concepts “*Person Name*” and “*Organisation Name*”.

2.1 Semantics of “Name”

Name schema is separated into a structural part (*xNL.xsd*) as shown in the XML schema diagram below and an “include” that contains enumerations used by the structural part (*xNL-types.xsd*). The structural part is expected to remain unchanged over the course of time while the “include” with enumerations may be easily changed to meet particular implementation needs.



The structure allows for different semantic levels based on the following paradigm:

- A simple data structure with minimum semantics should fit into the schema with minimal effort
- A complex data structure should fit into the schema without loss of any semantic information

Example 1 – no semantics

An imaginary database does not differentiate between a person and an organisation name with only one field allocated for storing the entire name information (unstructured data). This database can be mapped to xNL as follows:

```
<n:PartyName>
   <n:NameLine>Mr Jeremy Apatuta Johnson</n:NameLine>
</n:PartyName>
```

In this example, information related to party name, resides in *NameLine* element. It has no semantic information that may indicate what kind of name it is and what the individual name elements are (i.e., the data has not been parsed into first name, last name, title, etc.). What is known is that it is a name of some party, be it a person or an organisation.

Example 2 – minimal semantics

The next complexity level is when a database differentiates between person and organisation name. In this case, names can be placed in their respective places inside the structure.

Person name:

```
<n:PartyName>
   <n:PersonName>
          <n:NameElement>Mr Jeremy Apatuta Johnson</n:NameElement>
   </n:PersonName>
</n:PartyName>
```

This example shows that name information belongs to an individual, but the semantics of the individual name elements (e.g. What is “Mr”, “Jeremy”, etc.) are unknown.

Organisation name:

```
<n:PartyName>
   <n:OrganisationName>
          <n:NameElement>Khandallah Laundering Ltd.</n:NameElement>
   </n:OrganisationName>
</n:PartyName>
```

This example is similar to the previous one, except that the name belongs to an organisation.

Example 3 – full semantics

The next complexity level is when a database differentiates between person and organisation name and also differentiates between different name elements within a name. The data is structured.

```
<n:PartyName>
   <n:PersonName>
          <n:NameElement Abbreviation="true" ElementType="Title">Mr</n:NameElement>
          <n:NameElement ElementType="FirstName">Jeremy</n:NameElement>
          <n:NameElement ElementType="MiddleName">Apatuta</n:NameElement>
          <n:NameElement ElementType="LastName">Johnson</n:NameElement>
          <n:NameElement ElementType="GenerationIdentifier">III</n:NameElement>
          <n:NameElement ElementType="GenerationIdentifier">Junior</n:NameElement>
          <n:NameElement ElementType="Title">PhD</n:NameElement>
   </n:PersonName>
</n:PartyName>
```

This example introduces *ElementType* attribute that indicates the exact meaning of the name element. This is an additional level of semantics that is supported through enumerated values. Technically, the enumerations sit in a separate schema “include” (*xNL-types.xsd*).

An example of such enumeration is a list of name element types for a person name.

```
<xs:simpleType name="PersonNameElementsEnumeration">
   <xs:restriction base="xs:string">
          <xs:enumeration value="PrecedingTitle"/>
          <xs:enumeration value="Title"/>
          <xs:enumeration value="FirstName"/>
          <xs:enumeration value="MiddleName"/>
          <xs:enumeration value="LastName"/>
          <xs:enumeration value="OtherName"/>
          <xs:enumeration value="Alias"/>
          <xs:enumeration value="GenerationIdentifier"/>
   </xs:restriction>
</xs:simpleType>
```

These and other enumerations used in the CIQ Specifications are built using common sense and with a culture-specific view of the subject area (in this case Anglo-American culture), rather than adopted from a specific application. The reason why we say “cultural specific view” is because some cultures do not have the concept of FirstName, MiddleName and so on.

2.2 Data types

All elements and attributes in xNL schema have strong data types.

All free-text values of elements (text nodes) and attributes are constrained by simple type “*string*” defined in *xNL-types.xsd*. This type has a limit on the number of characters it may contain.

Other XML Schema data types are also used throughout the schema.

2.3 Enumerations

The *Name*, *Address* and *Party* schemas come with enumerations designed to satisfy common usage scenario, but there is always a possibility that a specific application requires enumerated values that are not part of the standard xNL specifications. It is acceptable for specific applications to provide their own enumerated values, but it is important that all participants involved in the data exchange with the application need to be aware of what the enumerated values are and that they are different from the ones provided by this specification to enable interoperability. Therefore, some agreement should be in place between the participants involved in the data exchange process where the enumerations have been customised to achieve better interoperability.

Example – point-to-point

Assume that participants of some data exchange agreed that for their purpose only a very simple name structure is required. One of the options for them is to modify *PersonNameElementsEnumeration* simple type in *xNL-types.xsd* file with the following values:

```
<xs:simpleType name="PersonNameElementsEnumeration">
   <xs:restriction base="xs:string">
          <xs:enumeration value="Title"/>
          <xs:enumeration value="FirstName"/>
          <xs:enumeration value="MiddleName"/>
          <xs:enumeration value="LastName"/>
   </xs:restriction>
</xs:simpleType>
```

Example – locale specific

In Russia, it would be more appropriate to use the following enumeration:

```
<xs:simpleType name="PersonNameElementsEnumeration">
   <xs:restriction base="xs:string">
          <xs:enumeration value="Title"/>
          <xs:enumeration value="Name"/>
          <xs:enumeration value="FathersName"/>
          <xs:enumeration value="FamilyName"/>
   </xs:restriction>
</xs:simpleType>
```

Again, it is up to the implementers to modify *PersonNameElementsEnumeration* simple type in *xNL-types.xsd* file.

2.4 Order of elements and presentation

Order of name elements should be preserved for correct presentation (e.g. printing name elements on a envelope).

If an application needs to present the name to a user it may not always be aware about the correct order of the elements if the semantics of the name elements are not available.

Example – normal order

```
Mr Jeremy Apatuta Johnson PhD
```

could be presented as follows

```
<n:PartyName>
   <n:PersonName>
          <n:NameElement>Mr</n:NameElement>
          <n:NameElement>Jeremy</n:NameElement>
          <n:NameElement>Apatuta</n:NameElement>
          <n:NameElement>Johnson</n:NameElement>
          <n:NameElement>PhD</n:NameElement>
   </n:PersonName>
</n:PartyName>
```

and restored back to *Mr Jeremy Apatuta Johnson PhD*.

Any other order of *NameElement* tags in the XML fragment could lead to an incorrect presentation of the name.

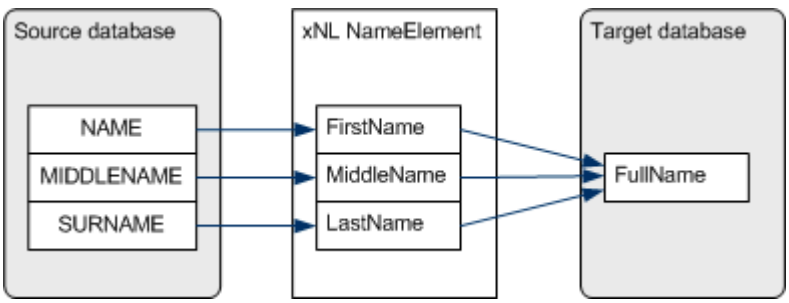
2.5 Data mapping

Mapping data between the xNL schema and a target database is not expected to be an issue as xNL provides enough flexibility for virtually any level of data decomposition. However, the main issue lies in the area of mapping a data provider with a data consumer through xNL. This may be a challenging task that requires additional name parsing software.

For example, consider a data provider that has a person name in one line (free text) and a data consumer that has a highly decomposed data structure for a person's name requires first name, family name and title to reside in their respective fields. There is no way of putting the provided data (free text) in the target data structure without parsing it first using some parsing tool. Such parsing is expected to be the responsibility of the data consumer.

Example – complex-to-simple mapping

The source database easily maps to the xNL *NameElement* qualified with *ElementType* attribute set to values as in the diagram



Source database

NAME	MIDDLENAME	SURNAME
John	Anthony	Jackson

xNL

```
<n:PersonName>
   <n:NameElement n:ElementType="FirstName">John</n:NameElement>
   <n:NameElement n:ElementType="MiddleName">Anthony</n:NameElement>
   <n:NameElement n:ElementType="LastName">Jackson</n:NameElement>
</n:PersonName>
```

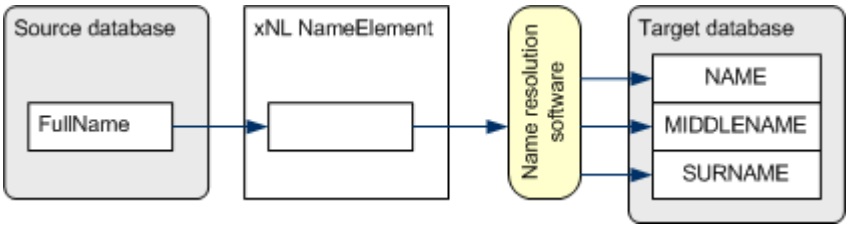
Target database

FULLNAME
John Anthony Jackson

This type of mapping does not present a major challenge as it is a direct mapping from source to xNL and then concatenating the data values to form the full name to be stored in a database field/column.

Example – simple-to-complex mapping

The source database has the name in a simple unparsed form which can be easily mapped to xNL, but cannot be directly mapped to the target database as in the following diagram:



Source database

FULLNAME
John Anthony Jackson

xNL

```
<n:PersonName>
   <n:NameElement>John Anthony Jackson</n:NameElement>
</n:PersonName>
```

At this point, the name resolution/parsing software splits *John Anthony Jackson* into a form acceptable by the target database.

Target database

NAME	MIDDLENAME	SURNAME
John	Anthony	Jackson

2.6 Data quality

xNL schema allows for data quality information to be provided as part of the entity using attribute *DataQuality* that can be set to either "*Valid*" or "*Invalid*", if such status is known. If *DataQuality* attribute is omitted, it is presumed that the validity of the data is unknown.

DataQuality attribute refers to the content of a container, e.g. *PersonName*, asserting that all the values are known to be true and correct. This specification has no provision for partial data quality where some parts of the content are correct and some are not or unknown.

Example – data quality

```
<n:PersonName n:DataQuality="Valid">
   <n:NameElement>John Anthony Jackson</n:NameElement>
</n:PersonName>
```

In this example *John Anthony Jackson* is known to be the true and correct value asserted by the sender of this data.

This feature allows the recipient of data to get an understanding of the quality of data they are receiving and thereby, assists them to take appropriate measures to handle the data according to its quality.

2.6.1 Data quality verification and trust

This specification does not mandate any data verification rules or requirements. It is entirely up to the data exchange participants to establish them.

Also, the participants need to establish if the data quality information can be trusted.

2.6.2 Data validation

This specification does not mandate any data validation rules or requirements. It is entirely up to the data exchange participants to establish such rules and requirements.

2.7 Extensibility

All elements in *Name*, *Address* and *Party* namespaces are extensible allowing for any number of attributes from a non-target namespace to be added.

All elements share the same declaration:

```
<xs:anyAttribute namespace="##other" processContents="lax"/>
```

Although this specification provides an extensibility mechanism, it is up to the participants of the data exchange process to agree on the use of any extensions to the target namespace.

This specification mandates that an application should not fail if it encounters an attribute from a non-target namespace. The application may choose to ignore or remove the attribute.

2.7.1 Practical applications

System-specific identifiers

Participants involved in data exchange may wish to add their system specific identifiers for easy matching of known data, e.g. if system A sends a message containing a name of a person to system B as in the example below

```
<n:PartyName xmlns:b="urn:acme.org:corporate:IDs" b:PartyID="123445">
   <n:PersonName>
          <n:NameElement>John Johnson</n:NameElement>
   </n:PersonName>
</n:PartyName>
```

then Attribute *b:PartyID="123445"* is not in xNL namespace and acts as an identifier for system A. When system B returns a response or sends another message and needs to include information about the same party, it may use the same identifier as in the following example:

```
<n:PartyName xmlns:b="urn:acme.org:corporate:IDs" b:PartyID="123445" />
```

The response could include the original payload with the name details.

Additional metadata

Sometime it is required to include some additional metadata that is specific to a particular system or application. Consider these examples:

```
<n:PartyName xmlns:x="urn:acme.org:corporate" x:OperatorID="buba7">
   ............
```

```
<n:PartyName xmlns:b="urn:acme.org:corporate ">
   <n:PersonName>
          <n:NameElement b:Corrected="true">John Johnson</n:NameElement>
   </n:PersonName>
</n:PartyName>
```

2.8 Linking and referencing

Names can be referenced internally (i.e. within some XML infoset that contains both referencing and referenced elements) through *xlink:href* pointing at an element with *xml:id* with a matching value.

External entities can also be referenced if they are accessible by the recipient via HTTP(s)/GET.

The following example illustrates *PartyName* elements that reference other *PartyName* elements that reside elsewhere, in this case outside of the document.

```
<a:Contacts
   xmlns:a="urn:acme.org:corporate:contacts"
   xmlns:n="urn:oasis:names:tc:ciq:xsdschema:xNL:3.0/20050427"
   xmlns:xlink="http://www.w3.org/1999/xlink">
   <n:PartyName xlink:href="http://example.org/party?id=123445" xlink:type="locator"/>
   <n:PartyName xlink:href="http://example.org/party?id=83453485" xlink:type="locator"/>
</a:Contacts>
```

This example presumes that the recipient of this XML fragment has access to resource *http://example.org/party* and that the resource returns *PartyName* element as an XML fragment of *text/xml* MIME type.

Usage of xLink attributes may slightly differ from the original xLink specification. See *CIQ TC Party Relationships Specification* for more information on using xLink with xNL. The xLink specification is available at http://www.w3.org/TR/xlink/.

Element *PartyName* can be either of type *locator* or *resource* in relation to xLink.

2.9 ID attribute

Attribute *ID* is used with complex type *PersonNameType* and elements *PersonName* and *OrganisationName*. This attribute allows unique identification of the collection of data it belongs to. The value of the attribute should be unique within the scope of the application of xNL. It is recommended that the value should be globally unique. The term 'globally unique' means a unique identifier that is "mathematically guaranteed" to be unique. For example, GUID (Globaly Unique Identifier) is a unique identifier that is based on the simple principle that the total number of unique keys (or) is so large that the possibility of the same number being generated twice is virtually zero.

This unique ID attribute should be used to uniquely identify collections of data as in the example below:

Application A supplies an xNL fragment containing some *PersonName* to Application B. The fragment contains attribute *ID* with some unique value.

```
<n:PartyName n:ID="52F89CC0-5C10-4423-B367-2E8C14453926">
   <n:PersonName>
          <n:NameElement>Max Voskob</n:NameElement>
   </n:PersonName>
   <n:OrganisationName>
          <n:NameElement>Khandallah Laundering Ltd.</n:NameElement>
   </n:OrganisationName>
</n:PartyName>
```

If *Application B* decides to reply to *A* and use the same xNL fragment it can only provide the outer element (*n:PartyName* in this case) with *ID* as the only attribute.

```
<n:PartyName n:ID="52F89CC0-5C10-4423-B367-2E8C14453926" />
```

Application *A* should recognise the value of *ID*, so no additional data is required from *B* in relation to this.

The exact behaviour of the *ID* attribute is not specified in this document and is left to the users to decide and implement.

The difference between the *ID* attribute and *xLink* attributes is that *ID* attribute cannot be resolved to a location of the data – it identifies already known data.

2.10 Schema customization guidelines

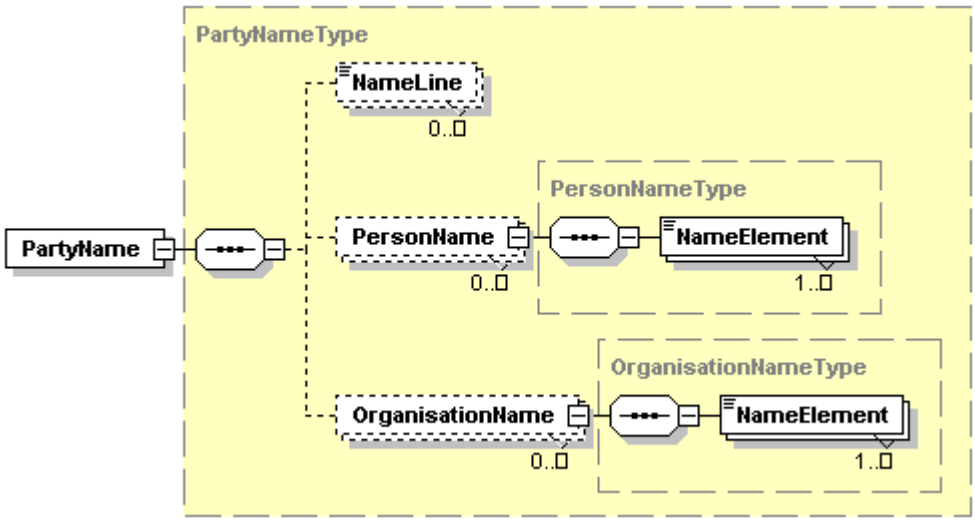
The broad nature and cultural diversity of entity "Name" makes it very difficult to produce one schema that would satisfy all applications and all cultures while keeping the size and complexity of the schema under control. This specification allows some changes to the schema by adopters of the schema to fit their specific requirements and constraints.

2.10.1 Namespace

The namespace identifier should be changed if it is possible for an XML fragment valid under the altered schema to be invalid under the original schema.

2.10.2 Reducing the structure

It is recommended to retain the minimum structure as in the following diagram:



This structure, although somewhat limited, still allows for most names to be represented, with exception for

- additional names (*KnownAs*), e.g. maiden name as part of *PartyName*
- organisation subdivision hierarchy (*SubdivisionName*), e.g. faculty / school / department

Any further reduction in structure may lead to loss of flexibility and expressive power of the schema.

It is not recommended to remove any attributes from the schema as they can be easily ignored during the processing.

2.10.3 Customizing the enumerations

Enumerations clarifying the meaning for generic elements (e.g. *NameElement*) were intentionally taken out of the main schema file into an include file (*xNL-types.xsd*) to make customisation easier.

The values of the enumerations can be changed or new ones added as required.

Proprietary enumeration example

Original xNL values for AliasTypeEnumeration	Possible proprietary values
MaidenName	MaidenName
NameChange	CommonUse
CommonUse	CivilName
	NickName
	PublishingName

The code for the new proprietary enumeration would look like this:

```
<xs:simpleType name="AliasTypeEnumeration">
   <xs:restriction base="xs:string">
          <xs:enumeration value="MaidenName"/>
          <xs:enumeration value="CommonUse"/>
          <xs:enumeration value="CivilName"/>
          <xs:enumeration value="NickName"/>
          <xs:enumeration value="PublishingName"/>
   </xs:restriction>
</xs:simpleType>
```

406 This level of flexibility allows some customization of the schema through changing the
407 enumerations only without changing the basic structure of the schema. It is important to
408 ensure that all schema users involved in data exchange use the same enumerations for
409 interoperability to be successful.

410 2.10.4 Implications

411 Any changes to the schemas are likely to break the compatibility one way or another.

412 It may be possible that an XML fragment created for the original schema is invalid for the
413 altered schema or vice versa. This issue needs to be considered before making any changes
414 to the schema and breaking the compatibility.

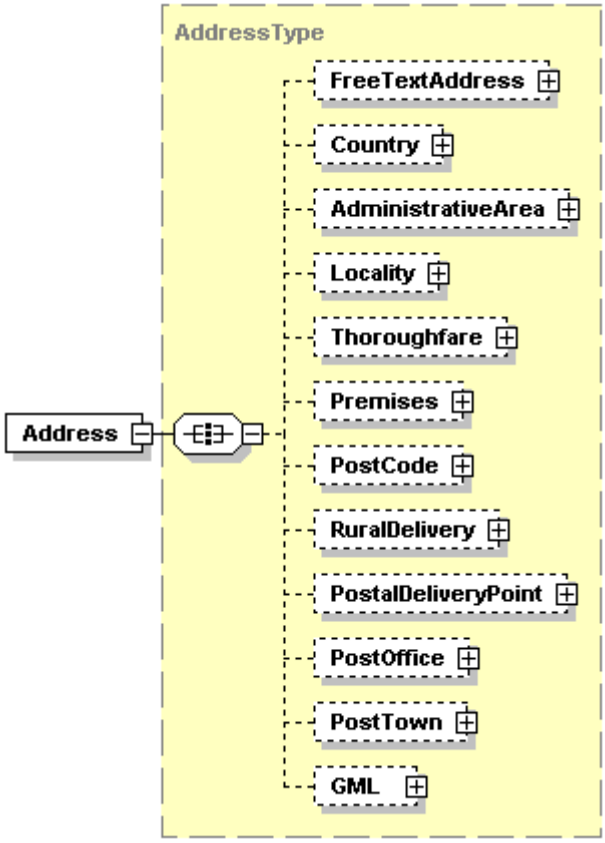
3 Entity “Address”

Entity “A*ddress*” has been modelled independent of any context as a standalone class to reflect some common understanding of concepts “*Location*” and “*Delivery Point*”.

The design concepts for “*Address*” are similar to “*Name*”. Refer to section 1.4 Common approaches for more information.

3.1 Semantics of “Address”

The high level schema elements of *xAL* schema are illustrated in the following diagram:



An address can be structured according to the complexity level of its source.

Example – free text

Suppose that the source database does not differentiate between different address elements and treats them as Address Line 1, Address Line 2, Address Line “N”, then the address information can be placed inside a free text container (element *FreeTextAddress*).

```
<a:Address>
   <a:FreeTextAddress>
          <a:AddressLine>Substation C</a:AddressLine>
          <a:AddressLine >17 James Street</a:AddressLine >
          <a:AddressLine>SPRINGVALE VIC 3171</a:AddressLine>
   </a:FreeTextAddress>
</a:Address>
```

It is up to the receiving application to parse this address and map it to the target data structure. It is possible that some sort of parsing software or human involvement will be required to accomplish the task.

Example – semi structured address

Assume that the address was captured in some semi-structured form such as State, Suburb and Street.

```
<a:Address>
   <a:AdministrativeArea>
          <a:Name>WA</a:Name>
   </a:AdministrativeArea>
   <a:Locality>
          <a:Name>OCEAN REEF</a:Name>
   </a:Locality>
   <a:Thoroughfare>
          <a:NameElement>16 Patterson Street</a:NameElement>
   </a:Thoroughfare>
</a:Address>
```

In this example, the free text information resides in containers that provide some semantic information on the content. E.g. State -> AdministrativeArea, Suburb -> Locality, Street -> Thoroughfare. At the same time, the Thoroughfare element contains street name and number in one line as free text, which may not be detailed enough for data structures where street name and number are separate fields.

Example – fully structured address

The following example illustrates an address structure that was decomposed into its atomic elements:

```
<a:Address>
   <a:AdministrativeArea>
          <a:Name a:Abbreviation="true" a:NameType="state">VIC</a:Name>
   </a:AdministrativeArea>
   <a:Locality>
          <a:Name>CLAYTON</a:Name>
          <a:SubLocality>Technology Park</a:SubLocality>
   </a:Locality>
   <a:Thoroughfare>
          <a:NameElement>Dandenong Road</a:NameElement>
          <a:Number a:EnumeratedType="RangeFrom">200</a:Number>
          <a:Number a:EnumeratedType="Separator">-</a:Number>
          <a:Number a:EnumeratedType="RangeTo">350</a:Number>
          <a:SubThoroughfare>
                 <a:NameElement>Fifth Avenue</a:NameElement>
          </a:SubThoroughfare>
   </a:Thoroughfare>
   <a:Premise>
          <a:NameElement>Toshiba Building</a:NameElement>
   </a:Premise>
   <a:PostalCode>
          <a:Number>3168</a:Number>
   </a:PostalCode>
</a:Address>
```

3.2 Geo-coordinates

Geo-coordinates can be provided by using Geography Markup Language (GML), an industry standard (http://www.opengis.net).

The reason for using some complex constructs from GML is due to the ambiguity of different coordinate systems, units and measurements. Also, GML incorporates a huge body of knowledge and expertise in geographical systems interoperability that can be reused for our purpose rather than re-inventing what has already been developed.

The content of *a:GML* must comply with the following requirements:

- Be from the GML namespace

- Refer to finest level of address details available in the address structure *a:GML* belongs to
- Be used unambiguously so that there is no confusion whether the coordinates belong to the postal delivery point (e.g. Post Box) or a physical address (e.g. flat) as it is possible to have both in the same address structure.

There is no restriction on the shape of the area *a:GML* can describe be it a point, polygon or some other object.

3.3 Data types

All elements and attributes in *xAL* schema have strong data types.

All free-text values of elements (text nodes) and attributes are constrained by simple type "*string*" defined in *xAL-types.xsd*. This type has a limit on the number of characters it may contain.

Other XML Schema defined data types are also used throughout xAL namespace.

3.4 Enumerations

Use of enumerations is identical to use of enumerations for entity "*Name*". Refer to section 2.3 Enumerations for more information.

Enumerations used in *xAL* reside in an "include" file *xAL-types.xsd*.

3.5 Order of elements and presentation

Order of address elements should be preserved for correct presentation in a fashion similar to what is described in section 2.4 Order of elements and presentation.

Child elements of *a:Address* can appear in any order as members of *xs:all* grouping as in the example below:

Example – order of second level elements in xAL

```
23 Archer Street       :      Thoroughfare
Chatswood, NSW 2067    :      Suburb, State, Post Code
Australia              :      Country
```

could be preserved and presented in XML as:

```
<a:Address>
   <a:Thoroughfare />
   <a:Locality />
   <a:AdministrativeArea />
   <a:PostCode />
   <a:Country />
</a:Address>
```

Some other elements can also appear in any order to preserve the original order.

3.6 Data mapping

Mapping data between *xAL* schema and a database is similar to that of entity "*Name*" as described in section 2.5 Data .

Example – normal order

```
23 Archer Street
Chatswood, NSW 2067
Australia
```

could be presented as follows

```
<a:Address>
   <a:FreeTextAddress>
          <a:AddressLine>23 Archer Street</a:AddressLine>
          <a:AddressLine>Chatswood, NSW 2067</a:AddressLine>
          <a:AddressLine>Australia</a:AddressLine>
   </a:FreeTextAddress>
</a:Address>
```

and restored back to

```
23 Archer Street
Chatswood, NSW 2067
Australia
```

during data formatting exercise.

Any other order of *AddressLine* tags in the XML fragment could lead to an incorrect presentation of the address.

3.7 Data quality

xAL schema allows for data quality information to be provided as part of the entity using attribute *DataQuality* as for entity "*Name*". Refer to section 2.6 Data for more information.

3.8 Extensibility

All element in *Address* namespace are extensible as described in section 2.7 Extensibility.

3.9 Linking and referencing

All linking and referencing rules described in section 2.8 Linking and apply to entity "*Address*".

Use of attribute ID is described in section 2.9 ID attribute.

3.10 Schema customization

Schema customisation rules and concepts described in section 2.10 Schema customization are fully applicable to entity "*Address*".

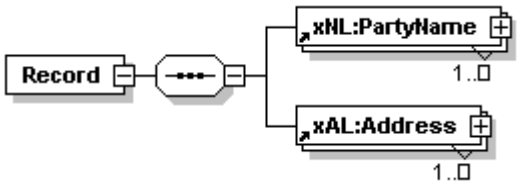
4 Combination of “Name” and “Address”

xNAL (*Name* and *Address*) schema is a container for combining related names and addresses. This specification recognises two ways of achieving so:

- Binding multiple names to multiple addresses (element *xnal:Record*)
- Binding multiple names to a single address for postal purposes (element *xnal:PostalLabel*)

4.1 Use of element xnal:Record

Element *xnal:Record* is a binding container that shows that some names relate to some addresses as in the following diagram:



The relationship type is application specific, but in general it is assumed that the people named in the xNL part somehow reside at the addresses specified in the xAL part. Use attributes from other namespace to specify the type of relationships and roles of names and addresses.

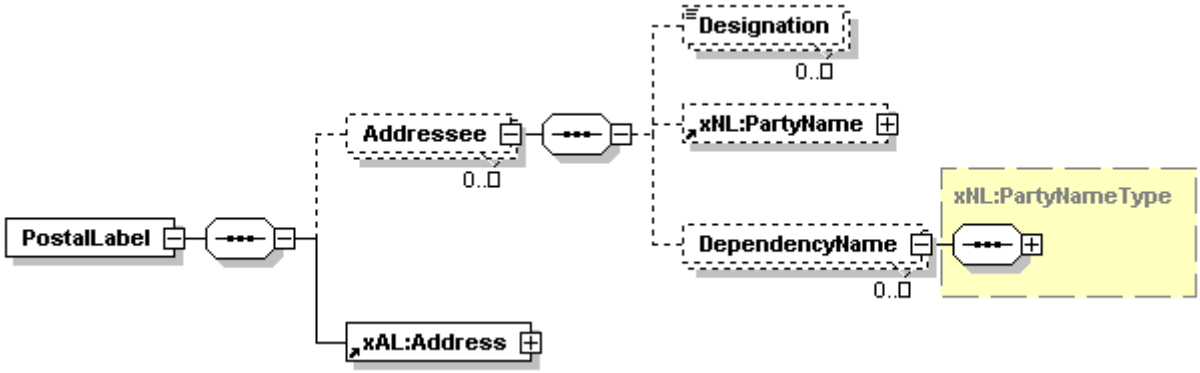
Example

Mr H G Guy, 9 Uxbridge Street, Redwood, Christchurch 8005

```
<xnal:Record>
   <n:PartyName>
          <n:NameLine>Mr H G Guy</n:NameLine>
   </n:PartyName>
   <a:Address>
          <a:Locality>
                 <a:Name>Christchurch</a:Name>
                 <a:SubLocality>Redwood</a:SubLocality>
          </a:Locality>
          <a:Thoroughfare>
                 <a:Number>9</a:Number>
                 <a:NameElement>Uxbridge Street</a:NameElement>
          </a:Thoroughfare>
          <a:PostCode>
                 <a:Identifier>8005</a:Identifier>
          </a:PostCode>
   </a:Address>
</xnal:Record>
```

4.2 Use of element xnal:PostalLabel

Element *xnal:PostalLabel* is a binding container that provides elements and attributes for information often used for postal / delivery purposes, as in the following diagram:



This structure allows for any number of recipients to be linked to a single address with some delivery specific elements such as *Designation* and *DependencyName*.

Example

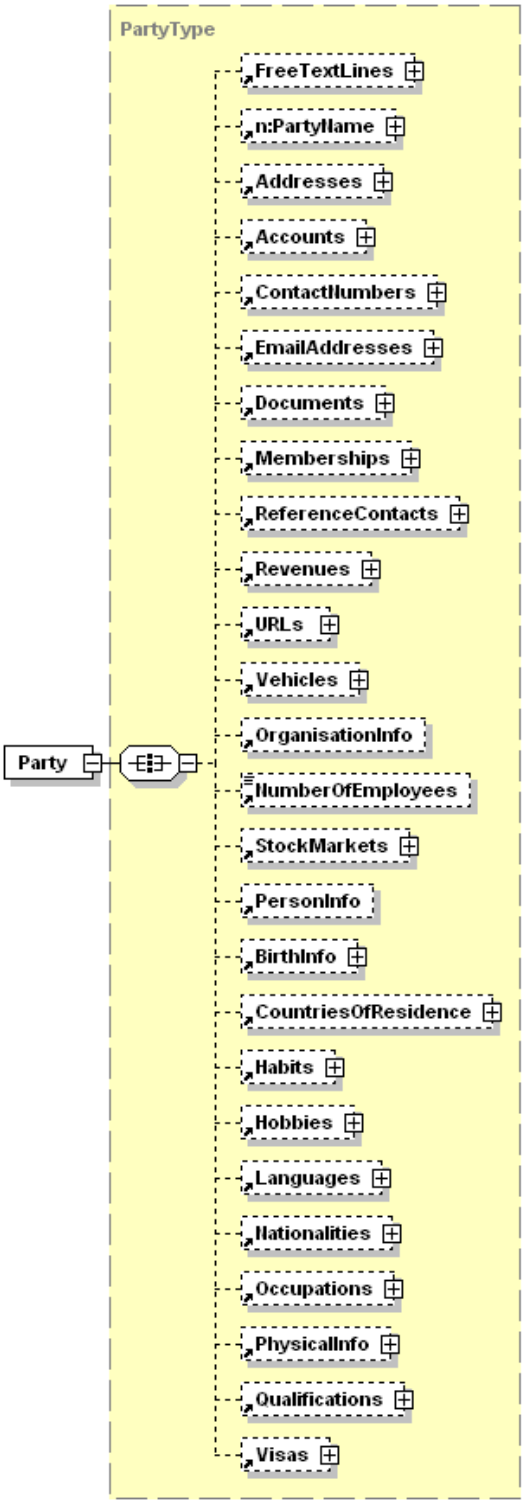
```
Attention: Mr S Mart
Name Plate Engravers
The Emporium
855 Atawhai Drive
Atawhai
Nelson 7001
```

Translates into the following xNAL fragment:

```
<xnal:PostalLabel>
   <xnal:Addressee>
          <xnal:Designation>Attention: Mr S Mart</xnal:Designation>
          <n:PartyName>
                 <n:NameLine>Name Plate Engravers</n:NameLine>
          </n:PartyName>
   </xnal:Addressee>
   <a:Address>
          <a:Locality>
                 <a:Name>Nelson</a:Name>
                 <a:SubLocality>Atawhai</a:SubLocality>
          </a:Locality>
          <a:Thoroughfare>
                 <a:NameElement>Atawhai Drive</a:NameElement>
                 <a:Number>855</a:Number>
          </a:Thoroughfare>
          <a:PostCode>
                 <a:Identifier>7001</a:Identifier>
          </a:PostCode>
   </a:Address>
</xnal:PostalLabel>
```

5 Entity “Party”

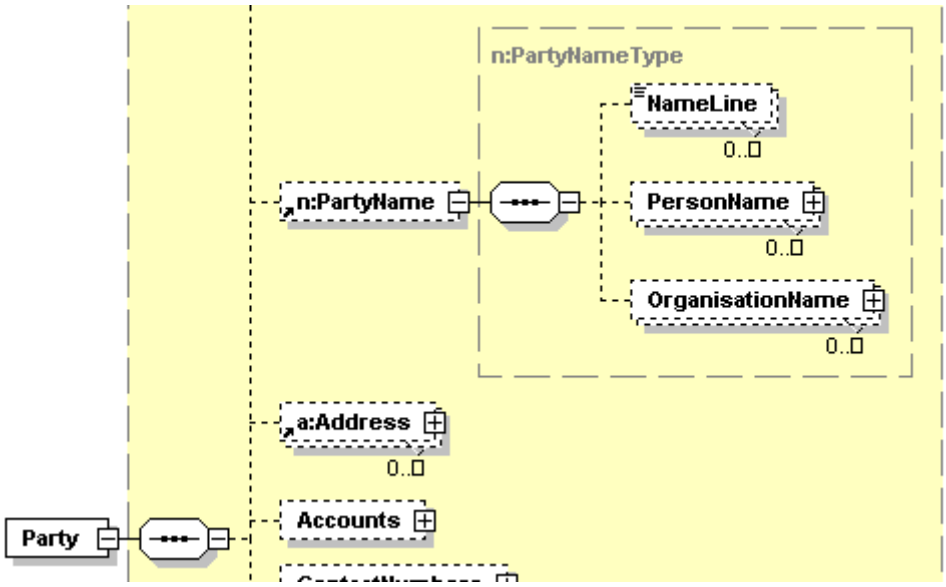
Entity “Party” encapsulates some most commonly used unique characteristics of *Person* or *Organisation*, such as name, address, personal details, contact details, body features, etc. The diagram below shows the high level structure of *Party*. The full schema can be found in *xPIL,xsd* file with enumerations in *xPIL-types.xsd* file. See the sample XML files for examples.



646 The schema consists of top level containers that may appear in any order or be omitted. The
647 containers are declared globally and can be reused by other schemas.

648 The shared elements apply to *organisation* as well as *person*.

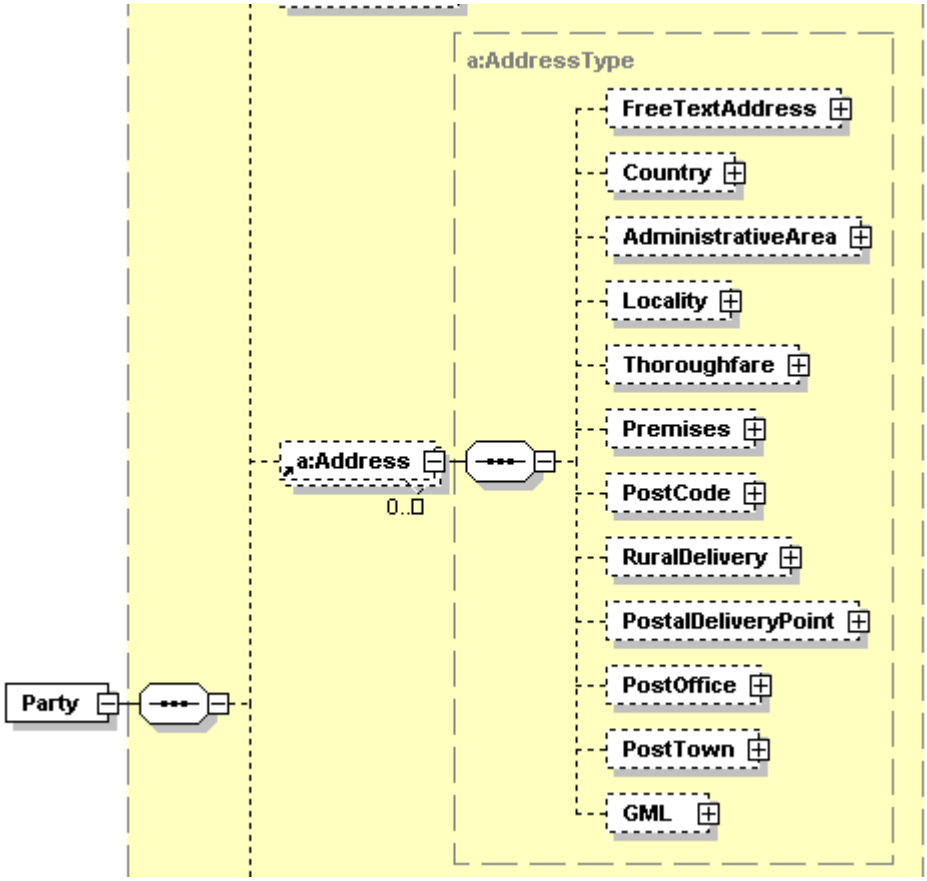
649 Name of the party reuses *PartyNameType* construct from xNL namespace as illustrated in the
650 following diagram:



651

652

653 Address of the party reuses *AddressType* construct from xAL namespace as illustrated in the
654 following diagram:



655

The design paradigm for this schema is similar to those of Name and Address entities. Likewise, it is possible to combine information at different detail and semantic levels.

The following example illustrates use of some of party constructs

Example – qualification details

```
<p:Qualifications>
   <p:Qualification>
          <p:QualificationElement
p:ElementType="QualificationName">BComp.Sc.</p:QualificationElement>
          <p:QualificationElement
p:ElementType="MajorSubject">Mathematics</p:QualificationElement>
          <p:QualificationElement
p:ElementType="MinorSubject">Statistics</p:QualificationElement>
          <p:QualificationElement p:ElementType="Award">Honours</p:QualificationElement>
          <p:InstitutionName>
                 <n:NameLine>University of Technology Sydney</n:NameLine>
          </p:InstitutionName>
   </p:Qualification>
</p:Qualifications>
```

Example – birth details

```
<p:BirthInfo p:BirthDateTime="1977-01-22T00:00:00"/>
```

Example – driver license

```
<p:Document p:ValidTo="2004-04-22T00:00:00">
   <p:IssuePlace>
          <a:Country>
                 <a:Name>Australia</a:Name>
          </a:Country>
          <a:AdministrativeArea>
                 <a:Name>NSW</a:Name>
          </a:AdministrativeArea>
   </p:IssuePlace>
   <p:DocumentElement p:ElementType="DocumentID">74183768C</p:DocumentElement>
   <p:DocumentElement p:ElementType="DocumentType">Driver License</p:DocumentElement>
   <p:DocumentElement p:ElementType="Priveledge">Silver</p:DocumentElement>
   <p:DocumentElement p:ElementType="Restriction">Car</p:DocumentElement>
</p:Document>
```

Example – contact phone number

```
<p:ContactNumber       p:MediaType="Telephone"       p:ContactNature="Business       Line"
p:ContactHours="9:00AM - 5:00PM">
   <p:ContactNumberElement p:ElementType="CountryCode">61</p:ContactNumberElement>
   <p:ContactNumberElement p:ElementType="AreaCode">2</p:ContactNumberElement>
   <p:ContactNumberElement p:ElementType="LocalNumber">94338765</p:ContactNumberElement>
</p:ContactNumber>
```

5.1 Dealing with Joint Party Names

xPIL schema represents details of a *Party*. The *Party* has a name as specified in *n:PartyName* element. A “Party” can be a unique name (e.g. A person or an Organisation) or a joint name (e.g. Mrs. Sarah Johnson and Mr. James Johnson (or) Mrs. & Mr. Johnson). In this case, all the other details of the party defined using xPIL apply to the party as a whole (i.e. to both the persons in the above example) and not to one of the Parties (e.g. say only to

Mrs. Sarah Johnson or Mr. James Johnson in the example). Also, all the addresses specified in *Addresses* element relate to the *Party* as a whole (i.e. applies to both Mrs. and Mr. Johnson in this example).

5.2 Data types

All elements and attributes in *xPIL* schema have strong data types.

All free-text values of elements (text nodes) and attributes are constraint by simple type "*string*" defined in *xPIL-types.xsd*. This type has a limit on the number of characters it may contain.

Other XML Schema defined data types are also used throughout the schema.

5.3 Enumerations

Use of enumerations is identical to use of enumerations for entity "*Name*". Refer to section 2.3 Enumerations for more information.

Enumerations used in *xPIL* reside in an "include" *xPIL-types.xsd*.

5.4 Order of elements and presentation

Order of elements without qualifier (@...type attribute) should be preserved for correct presentation in a fashion similar to what is described in section 2.4 Order of elements and presentation.

5.5 Data mapping

Mapping data between *xPIL* schema and a database is similar to that of entity "*Name*" as described in section 2.5 Data .

5.6 Data quality

xPIL schema allows for data quality information to be provided as part of the entity using attribute *DataQuality* as for entity "*Name*". Refer to section 2.6 Data for more information.

5.7 Extensibility

All element in *Party* namespaces are extensible as described in section 2.7 Extensibility.

5.8 Linking and referencing

All linking and referencing rules described in section 2.8 Linking and apply to entity "*Party*".

The following example illustrates *PartyName* elements that reference other *PartyName* element that resides elsewhere, in this case outside of the document.

```
<a:Contacts xmlns:a="urn:acme.org:corporate:contacts">
   <xnl:PartyName xlink:href="http://example.org/party?id=123445"/>
   <xnl:PartyName xlink:href="http://example.org/party?id=83453485"/>
</a:Contacts>
```

This example presumes that the recipient of this XML fragment has access to resource "*http://example.org/party*" (possibly over HTTP/GET) and that the resource returns as *PartyName* element as an XML fragment of *text/xml* MIME type.

Use of attribute ID is described in section 2.9 ID attribute.

5.9 Schema customization

Schema customisation rules and concepts described in section 2.10 Schema customization are fully applicable to entity "*Party*".

6 Miscellaneous

6.1 Documentation

Although, all schema files are fully documented using XML Schema annotations it is not always convenient to browse the schema itself. This specification is accompanied by a set of HTML files auto generated by XML Spy. Note that not all information captured in the schema annotation tags is in the HTML documentation.

6.2 Examples

Several examples of instance XML documents for name, address and party schemas are provided as XML files. The examples are informative and demonstrate the application of this Technical Specification.

The example files and their content are being constantly improved and updated on no particular schedule.

6.3 Contributions from Public

OASIS CIQ TC is open in the way it conducts its business. We welcome contributions from public in any form. Please, use “Send A Comment” feature on CIQ TC home page (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ciq) to tell us about:

- errors, omissions, misspellings in this specification, schemas or examples
- your opinion in the form of criticisms, suggestions, comments, etc
- willingness to contribute to the work of CIQ TC by becoming a member of the TC
- willingness to contribute indirectly to the work of CIQ TC
- provision of sample data that can be used to test the specifications
- implementation experience
- etc.

Appendix A. Notices