Customer Information Quality Specifications Version 3.0

Name (xNL), Address (xAL) and Party (xPIL)

Public Review Draft 02

15 June 2007

Specification URIs:

This Version:

http://docs.oasis-open.org/ciq/v3.0/prd02/specs/ciq-specs-v3-prd2.html
http://docs.oasis-open.org/ciq/v3.0/prd02/specs/ciq-specs-v3-prd2.doc
http://docs.oasis-open.org/ciq/v3.0/prd02/specs/ciq-specs-v3-prd2.pdf

Previous Version:

N/A

Latest Version:

http://docs.oasis-open.org/ciq/v3.0/specs/ciq-specs-v3.html
http://docs.oasis-open.org/ciq/v3.0/specs/ciq-specs-v3.doc
http://docs.oasis-open.org/ciq/v3.0/specs/ciq-specs-v3.pdf

Technical Committee:

OASIS Customer Information Quality TC

Chair:

Ram Kumar (kumar.sydney@gmail.com)

Editor:

Ram Kumar (kumar.sydney@gmail.com)

Related work:

This specification replaces or supercedes:

- OASIS CIQ extensible Name Language (xNL) V2.0 Committee Specification
- OASIS CIQ extensible Address Language (xAL) V2.0 Committee Specification
- OASIS CIQ extensible Name and Address Language (xNAL) V2.0 Committee Specification
- OASIS CIQ extensible Customer Information Language (xCIL) V2.0 Committee Specification

Declared XML Namespace(s):

urn:oasis:names:tc:ciq:3.0

Abstract:

This Technical Specification defines the OASIS Customer Information Quality Specifications Version 3.0 namely, Name (xNL), Address (xAL), Name and Address (xNAL) and Party Information (xPIL) specifications.

Status:

This document was last revised or approved by the OASIS CIQ TC on the above date. The level of approval is also listed above. Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/ciq/

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (www.oasis-open.org/committees/ciq/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/ciq/.

Notices

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", “CIQ”, “xNL”, “xAL”, xNAL”, “xPIL”, “xPRL”, “xCIL”, “xCRL” , “Genericode”, and “UBL” are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

1 Name, Address and Party

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in [RFC2119].

While RFC2119 permits the use of synonyms, to achieve consistency across specifications, “MUST” is used instead of “SHALL” and “REQUIRED”, “MUST NOT” instead of “SHALL NOT”, and “SHOULD” instead of “RECOMMENDED” in this specification. To enable easy identification of the keywords, uppercase is used for keywords.

1.2 Definitions

Following are the core entities and its definitions used by CIQ TC:

Name

Name of a person or an organization

Address

A physical location or a mail delivery point

Party

A Party could be of two types namely,

- Person
- Organization

An Organization could be a company, association, club, not-for-profit, private firm, public firm, consortium, university, school, etc.

Party data consists of many attributes (e.g. Name, Address, email address, telephone, etc) that are unique to a party. However, a person or organization’s name and address are generally the key identifiers (but not necessarily the unique identifiers) of a “Party”. A “Customer” is of type “Party”.

2 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. The *xNAL* schema design concept varies slightly as it is only a simple container for associating names and addresses.

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

2.1 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 Enumerations	Defines a set of enumerations to support Name entity
xAL.xsd	Entity Address	Defines a set of reusable types and elements for an address, location name or description
xAL-types.xsd	Entity Address Enumerations	Defines a set of enumerations to support address entity
xNAL.xsd	Name and Address binding	Defines two constructs to bind names and addresses for data exchange or postal purposes
xNAL-types.xsd	Name and Address binding Enumerations	Defines a set of enumerations to support name and address binding
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
xPIL-types.xsd	Entity Party (organisation or individual) Enumerations	Defines a set of enumerations to support party information entity
CommonTypes.xsd	Common Data Types and Enumerations	Defines a set of commonly used data types and enumerations in the CIQ Schemas
xLink-2003-12-31.xsd	xLink attributes	Defines a subset of xLink attributes as XML schema
*.gc files	Entity Party, Name, and Address	Defines a set of enumerations/code lists in genericode

2.2 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

2.3 Major CIQ Specification 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 basic CIQ specification entities:

- Name (Person or Organisation - see xNL.xsd)
- Address (see xAL.xsd)
- Name and Address combined (see xNAL.xsd)
- Personal details of a person (see xPIL.xsd)
- Organisation specific details (see xPIL.xsd)
- Party Relationships (see xPRL.xsd [not available in this release] and xLink-2003-12-31-revised.xsd)

These major entities are supported by code lists to add “semantics” to the data they represent. We categorise the major entities of CIQ Specifications into three namely,

- Name
- Address, and
- Party Centric Information

2.4 Common Design Concepts used in the Specifications

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 design concepts used in the CIQ Specifications (e.g. using code lists, customizing CIQ entity schemas, extending CIQ entity schemas, referencing between entities, defining business rules to constrain CIQ entity schemas) are equally applicable for all key entities of CIQ specifications namely, Name, Address and Party. These common concepts are explained in detail in section 3 (Entity “Name”). Users SHOULD study that section in detail before proceeding to other entities namely, Address and Party, as these concepts are applicable to these entities also.

2.5 Namespaces

Entity	Namespace	Suggested 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 xNAL-types.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-2003-12-31.xsd

2.6 Other Industry Specifications Used

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/ . The CIQ TC strongly recommends readers to read the xLink specification from W3C if they want to use this supported feature in CIQ Specifications.

This document contains references to Code List version 1.0, OASIS Committee Specification, May 2007 at http://www.oasis-open.org/committees/codelist. The CIQ TC strongly recommends readers to read the code list specification if they want to use this supported feature in CIQ Specification.

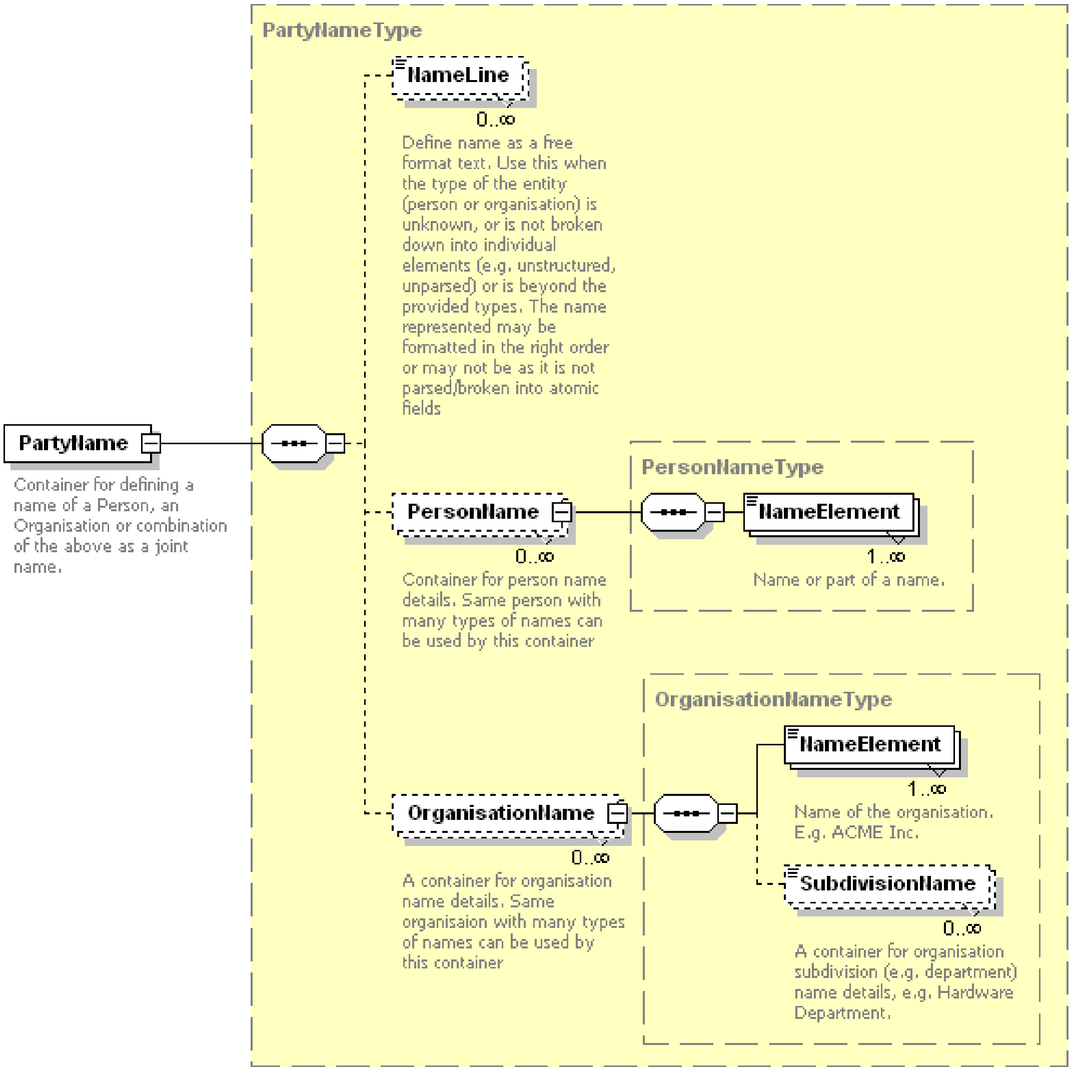
GeoRSS 2.0 (georss.org) from Open Geospatial Consortium (http://www.opengeospatial.net) has been referenced in this specification as it is critical to assuring interoperability with a variety of geospatial technologies, such as GIS, Spatial Data Infrastructures, Location Services, and the GeoWeb.

3 Entity “Name” (extensible Name Language)

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*”.

3.1 Semantics of “Name”

Name schema is separated into two parts: a structural part (*xNL.xsd*) as shown in the XML schema diagram below and separate enumeration/code list files (code lists defined in an XML schema (*xNL-types.xsd*) and also, code lists represented in genericode format as *.gc* files) supporting the structure. “Genericode” will be discussed in later sections. The structural part (*xNL.xsd*) SHOULD remain unchanged over the course of time while the code list/enumeration files (*xNL-types.xsd* and *.gc* files) MAY be easily changed to meet particular implementation needs.


PartyNameType
NameLine
0..∞
Define name as a free format text. Use this when the type of the entity (person or organisation) is unknown, or is not broken down into individual elements (e.g. unstructured, unparsed) or is beyond the provided types. The name represented may be formatted in the right order or may not be as it is not parsed/broken into atomic fields
PartyName
Container for defining a name of a Person, an Organisation or combination of the above as a joint name.
PersonNameType
PersonName
0..∞
Container for person name details. Same person with many types of names can be used by this container
NameElement
1..∞
Name or part of a name.
OrganisationNameType
NameElement
1..∞
Name of the organisation. E.g. ACME Inc.
OrganisationName
0..∞
A container for organisation name details. Same organisaion with many types of names can be used by this container
SubdivisionName
0..∞
A container for organisation subdivision (e.g. department) name details, e.g. Hardware Department.

In the schema structure above (*xNL.xsd*), “NameElement” stores the name of a party and the supporting enumeration lists referenced as *attributes* in the schema structure (see the *xNL.xsd* schema for the list of attributes or the HTML documentation of the schema), provide the semantic meaning of the data.

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

3.1.1 Example 1 – No Semantics (Unstructured/Free Text Data)

A typical database does not differentiate between a person and an organisation name where only one field has been 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. This is the maximum level of complexity. Data in this free formatted text form is classified as “poor quality” as it is subject to different interpretations of the data and will cause interoperability problems.

Many common applications fall under this “No Semantics” category.

3.1.2 Example 2 – Minimal Semantics (Partially Structured Data)

The medium level of complexity is when a database differentiates between person and organisation name. In this case, names are placed in the appropriate elements namely, *PersonName* or *OrganisationName* 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 are the meanings of “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.

Many common applications fall under this of “Minimal Semantics” category.

3.1.3 Example 3 – Full Semantics (Fully Structured Data)

The maximum level of complexity 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. Few applications and in particular, applications dealing with data quality and integrity, fall in this “Full Semantics” category and often, the database supported by these applications are high in the quality of the data it manages. This is an additional level of semantics that is supported through code list/enumerated values. Technically, the enumerations sit in a separate schema (*xNL-types.xsd*) or in genericode files.

An example of enumeration is a list of name element types for a person name defined in *xNL-types.xsd* is shown below.

```
<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>
```

3.2 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 a simple type “*String*” (255 characters in size and collapsed white spaces) defined in *CommonTypes.xsd*. Other XML Schema data types are also used throughout the schema.

3.3 Code Lists (Enumerations)

This is an important section that users MUST pay more attention.

3.3.1 What is a Code List?

A *code list* (also called *enumeration*) defines a classification schema and a set of classification values to support the scheme. For example, “Administrative Area” is a classification scheme and a set of classification values for this classification scheme could be: State, City, Province, Town, Region, District, etc.

XML Schema describes the structural and lexical constraints on an XML document. Some information items in a document are described in the schema lexically as a simple value whereby the value is a code representing an agreed-upon semantic concept. The value used is typically chosen from a set of unique coded values enumerating related concepts. These sets of coded values are sometimes termed *code lists*.

3.3.2 The importance of Code Lists for CIQ Specifications

Earlier versions of CIQ Name, Address and Party Information specifications had concrete schema grammar to define the party entities. This did not satisfy many name, address and party data usage scenarios that are geographic and cultural specific. For example, in certain countries, the concept of first name, middle name, and last/family/surname does not exist. Representing names from these countries in the earlier version of CIQ Specifications were difficult as its name schema (for example, v2.0 of *xNL.xsd*) had element names as *FirstName, MiddleName*, and *LastName,* and they were semantically incorrect metadata for the data in this example. To be precise, in a country where the concept of *First Name* does not exist, using *First Name* element of CIQ specification was semantically incorrect. Therefore, the use of enumeration lists/code lists approach (that is customisable) in this version of the specifications provides the correct semantics to the data.

Let us look at the following example:

```
<n:PartyName>
   <n:PersonName>
          <n:NameElement ElementType="FatherName">Venkat</n:NameElement>
          <n:NameElement ElementType="GivenName">Ramkumar</n:NameElement>
   </n:PersonName>
</n:PartyName>
```

In the above example, *NameElement* is the XML tag used to represent name data and it is irrelevant whether the name is a *First Name*, *Middle Name* or *Last Name* or *Surname*. The attribute *ElementType* that uses enumeration list of semantic information for a name provides the semantics to the data (*FatherName* and *GivenName* in this example) represented in *NameElement.*

3.3.3 Customisable Code Lists

The *Name*, *Address* and *Party* schemas in this version provides code lists/enumerations designed to satisfy common usage scenarios of the data by providing semantically correct metadata to the data. These code lists are customisable to satisfy different name and address data requirements, but at the same time ensures that the core CIQ schema structure is intact i.e., there is no need to change the schema to suit specific requirements. A default set of code list/enumerated values are provided with the schemas and these default values are not complete by any means and therefore, is customisable by the user to suit their requirements.

The default code list values/enumerations used in the CIQ Specifications are built using common sense and with culture-specific view of the subject area (in this case Anglo-American culture, where the terms such as *First Name*, *Middle Name*, *Last Name* are used), 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 *First Name, Middle Name*, and *Last Name* and so on.

NOTE: The code list/enumeration values for different code/enumeration lists that are provided as part of the specifications are not complete. They only provides some sample values and it is up to the end users to customise them to meet their data exchange requirements if the default values are incomplete, not appropriate or over kill

There is always a possibility that a specific application requires certain enumerated values that are not part of the standard *xNL, xAL* and *xPIL* specifications. It is acceptable for specific applications to provide their own enumerated values (e.g. could be new one, delete an existing default one), but it is important that all participants (could be internal business systems or external systems) involved in data exchange SHOULD be aware of what the new enumeration values are to enable interoperability. Therefore, some agreement SHOULD be in place between the participants involved in the data exchange process (e.g. Information Exchange Agreement for data exchange) to agree on the agreed enumeration values. These agreed enumeration values SHOULD also be governed to manage any changes to the agreed enumeration values in order to prevent interoperability breakdown. Any further information about these sorts of agreements is outside the scope of the CIQ technical committee.

If there is neither a requirement to use the customisable enumeration list nor the default enumeration values provided with the specifications, make the list empty, but remember that the semantic meaning of the data will be lost.

3.3.4 Improving Interoperability using Code Lists

Using customisable code list approach provided by CIQ Specifications, interoperability of data (represented using CIQ Specifications) between applications can be significantly improved. Any attribute/element that can add semantic meaning to a piece of data (e.g. type of address, where the value “Airport” adds semantics to an address data) is defined as a customisable code list in CIQ Specifications. For example, *PersonName* element in *xNL.xsd* uses an attribute *PersonIDType* that provides a default code list, but with no default values. When a code list has no values, XML Parsers treat the attribute/element that references the code list as XML schema data type *string*. This allows an application to define any string value without any restriction. This could result in interoperability breakdown between the sending application and the receiver application. To improve interoperability by controlling the use of the values for the string, users SHOULD specific values in the code list that SHOULD agreed by the parties exchanging the data. As a result, applications and users can be confident that the data that is exchanged conforms to the code list values that have been agreed.

To provide enough flexibility to users to define the semantics of the data, over 90 default code lists (most of them are empty, i.e., no default code values are provided) are provided by CIQ Specifications that are customisable by users to improve interoperability.

3.4 Using Code Lists in CIQ Specifications – Two Options

CIQ Specifications provide TWO OPTIONS to users to define and manage code lists. The options are:

- **OPTION 1:** An XML schema file per CIQ entity (*Name, Address* and *Party*) representing all code lists for the entity. The files are *xNL-types.xsd* (for *Name* Entity code lists), *xAL-types.xsd* (For *Address* Entity code lists), *xNAL-types.xsd* (for *Name* and *Address* Entities code list) and *xPIL-types*.xsd (for *Party* Entity code lists). This is the "DEFAULT" approach for using code lists.
- **OPTION 2:** A genericode based code list file (.gc) per code list for all CIQ entities (*Name, Address* and *Party*). Genericode is an OASIS industry specification for representing, validating, and managing code lists. For example, *xNL-types.xsd* has 10 code lists in Option 1, is represented as 10 individual genericode (.gc) files in this option. This option does not use xNL-types.xsd, xAL-types.xsd, xNAL-types.xsd, and xPIL-types.xsd Code List schemas.

Users MUST choose one of the above two options as part of the specification implementation, but not both.

3.4.1 Why Two Options

Option 2 (Genericode approach) uses two pass validations on a CIQ XML document instance (first pass for XML document structural and lexical validation against the core CIQ XML schema (*xNL.xsd*) and second pass for validation of code list value in the XML document).

If only this option is provided as part of the CIQ specifications, end users implementing CIQ XML schema that is included as part of their application specific schema to represent party data, will be forced to perform two pass validation on the application's XML document instance and in particular, the fragments in the XML document where party data is represented using CIQ, because their application schema MAY not support code lists in genericode. This limits the usage of CIQ specifications and hence, two options are provided to enable end users to pick an approach that suits their requirements. The two options are explained in the following sections.

3.4.2 Option 1 – "Include" Code Lists (The Default Approach)

"Include" code lists are XML schemas that are "included" in the CIQ entity structure XML schemas, i.e., *xNL.xsd* (Name Entity schema) "includes" *xNL-types.xsd* code list schema, *xAL.xsd* (Address Entity schema) "includes" *xAL-types.xsd* code list schema, *xNAL.xsd* schema "includes" *xNAL-types.xsd* code list schema, and *xPIL.xsd* (party entity schema) "includes" *xPIL-types.xsd* schema.

Users MAY modify the code list XML schemas to add or delete values depending upon their data exchange requirements without modifying the structure of the CIQ entity schemas. Validation of the code list values will be performed by XML parsers as part of the XML document instance validation in "one" pass (i.e., XML document structure validation and the code list value validation will be performed in one pass). Any changes to the code list schema results in changes to the software code (e.g. java object generated from *xNL.xsd* using XML Beans must be re-created) based on the entity schema as the entity schema "includes" the code list schema.

The code list values for code lists provided as part of CIQ Specifications v3.0 are only sample values and by no means are accurate or complete list of values. It is up to the users to customise the default code list. However, when exchanging data with more than one party (trading partner or application), it is important that all the concerned parties SHOULD be aware of the code list and the values that will be used as part of the data exchange process to ensure interoperability.

3.4.2.1 Representing Code Lists (Option 1)

Code Lists for each CIQ entity (Name, Address and Party) are represented in one XML schema file per CIQ entity. For example, *xNL-types.xsd* represents 10 code lists.

In some cases, the code list/enumeration list values are empty, i.e. no default values are provided. Under this circumstance, the attribute that uses this empty enumeration is of xml schema data type *string* and users can use any value.

For those attributes that do not use any enumeration list and is of xml schema type *string,* users SHOULD ensure that any value that is defined for the attribute that adds semantic value to the attribute's associated element's data SHOULD be agreed between all the parties that are involved in the data exchange. For example, the element *PartyName* has a child element called *NameLine* that is used to represent free format name. This child element has an attribute called *Type* that does not use an enumeration list and is of xml schema data type *string*. If users decide to use this attribute with values that add semantics to the data represented in *NameLine*, say, *FullName* or *PartialName*, users SHOULD agree to these values with the parties involved in data exchange to ensure interoperability of data between the parties.

3.4.2.1.1 Code List Representation (Option 1) – An Example

The following example shows an XML schema representation of code list for *SubDivisionTypeList* provided by CIQ specification as part of *xNL-types.xsd*.

```
<xs:simpleType name=SubDivisionTypeList”>
   <xs:annotation>
          <xs:documentation> A list of common types for sub divisions
          </xs:documentation>
   </xs:annotations>
   <xs:restriction base=”xs:string”>
          <xs:enumeration value=”Department”/>
          <xs:enumeration value=”Branch”/>
          <xs:enumeration value=”Business Unit”/>
          <xs:enumeration value=”School”/>
          <xs:enumeration value=”Section”/>
   </xs:restriction>
</xs:simpleType>
```

3.4.2.2 Customizing Code Lists (Option 1)

Meeting all requirements of different cultures and ethnicity in terms of representing the names in one specification is not trivial. This is the reason why code lists/enumerations are introduced in order to keep the specification/schema simple, but at the same time provide the flexibility to adapt to different requirements.

Code lists 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 MAY be changed or new ones added as required.

NOTE: The code lists values for different enumeration lists that are provided as part of the specification are not complete. They only provides some sample values and it is up to the end users to customise them to meet their data exchange requirements if the default values are incomplete, not appropriate or over kill

3.4.2.3 End User Customised Code List (Option 1) – An Example

In the following example, the code list "*OrganisationNameTypeList*" under "*xNL-types.xsd*" is customised by replacing the default values with new values to meet user requirements.

Original xNL values for OrganisationNameTypeList	Possible end user customised values
LegalName	ReportedName
NameChange	OriginalName
CommonUse	LegalName
PublishingName	
OfficialName	
UnofficialName	
Undefined	

The code for the specification provided original code list would look like the following:

```
<xs:simpleType name="OrganisationNameTypeList">
   <xs:restriction base="xs:string">
          <xs:enumeration value="LegalName"/>
          <xs:enumeration value="NameChange"/>
          <xs:enumeration value="CommonUse"/>
         <xs:enumeration value="PublishingName"/>
          <xs:enumeration value="OfficialName"/>
          <xs:enumeration value="UnofficialName"/>
         <xs:enumeration value="Undefined"/>
   </xs:restriction>
</xs:simpleType>
```

The code for the new end user customised code list would look like the following:

```
<xs:simpleType name="OrganisationNameTypeList">
   <xs:restriction base="xs:string">
          <xs:enumeration value="ReportedName"/>
          <xs:enumeration value="OriginalName"/>
          <xs:enumeration value="LegalName"/>
   </xs:restriction>
</xs:simpleType>
```

This level of flexibility allows customization of the *xNL.xsd* schema through changing the code lists only, without changing the basic structure of the *xNL.xsd* schema. It is important to ensure that all schema users involved in data exchange SHOULD use the same code lists for interoperability to be successful. This SHOULD be negotiated between the data exchange parties and a proper governance process SHOULD be in place to manage this process.

3.4.2.4 Code List Use (Option 1) Example – Point-to-Point

Assume that participants of a data exchange agreed that for their purpose only a very simple name structure is required. One of the options for them is to modify *PersonNameElementsList* simple type in the *xNL-types.xsd* file with the following values and remove the rest of the default values provided by the specification:

```
<xs:simpleType name="PersonNameElementsList">
   <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>
```

3.4.2.5 Code List Use (Option 1) Example – Locale Specific

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

```
<xs:simpleType name="PersonNameElementList">
   <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 involved in data exchange to modify *PersonNameElementList* simple type in *xNL-types.xsd* file.

3.4.3 Option 2 – Code Lists using Genericode Approach

Option 1 is the default approach for CIQ Specifications to use code lists. However, users are given the choice to use Option 2 instead of Option 1. It is up to the users to decide which approach to use and this is based on their requirements.

The OASIS Code List Representation format, "*Genericode*", is a single industry model and XML format (with a W3C XML Schema) that can encode a broad range of code list information. The XML format is designed to support interchange or distribution of machine-readable code list information between systems. Details about this specification are available at: http://www.oasis-open.org/committees/codelist.

Let us consider an instance where trading partners who use CIQ Specifications for exchanging party related data. The trading partners MAY wish to agree that different sets of values from the same code lists MAY be allowed at multiple locations within a single document (perhaps allowing the state for the buyer in an order is from a different set of states than that allowed for the seller). Option 1 approach MAY not be able to accommodate such differentiation very elegantly or robustly, or possibly could not be able to express such varied constraints due to limitations of the schema language's modelling semantics. Moreover it is not necessarily the role of CIQ Entity schemas to accommodate such differentiation mandated by the use of it. Having a methodology and supporting document types with which to perform code list value validation enables parties involved in document exchange to formally describe the sets of coded values that are to be used and the document contexts in which those sets are to be used. Such a formal and unambiguous description SHOULD then become part of a trading partner contractual agreement, supported by processes to ensure the agreement is not being breached by a given document instance.

This option uses a "two" pass validation approach, whereby, the "first" pass validation, allows the XML document instance to be validated for its structure and well-formedness (ensures that information items are in the correct places and are correctly formed) against the entity XML schema, and the "second" pass validation allows the code list values in XML document instance to be validated against the genericode based code lists and this does not involve the entity schemas. Any change to the genericode based code list does not require changes to the software code (e.g. java object must be re-created) based on the entity schema as the entity schema has nothing to do with the genericode based code list.

3.4.3.1 Code List (Option 2) Value Validation Methodology

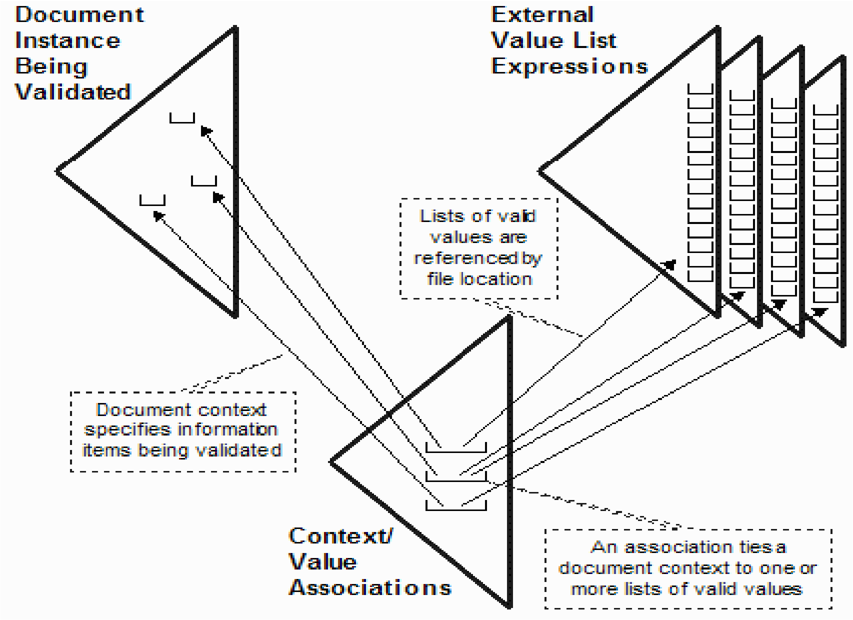
OASIS Codelist Technical Committee describes a methodology for validating code list values and supporting document types with which trading partners can agree unambiguously on the sets of code lists, identifiers and other enumerated values against which exchanged documents must validate.   The objective of applying this methodology to a set of document instances being validated is to express the lists of values that are allowed in the contexts of information items found in the instances. One asserts that particular values must be used in particular contexts, and the validation process confirms the assertions do not fail.

3.4.3.1.1 Two Pass Value Validation (Option 2)

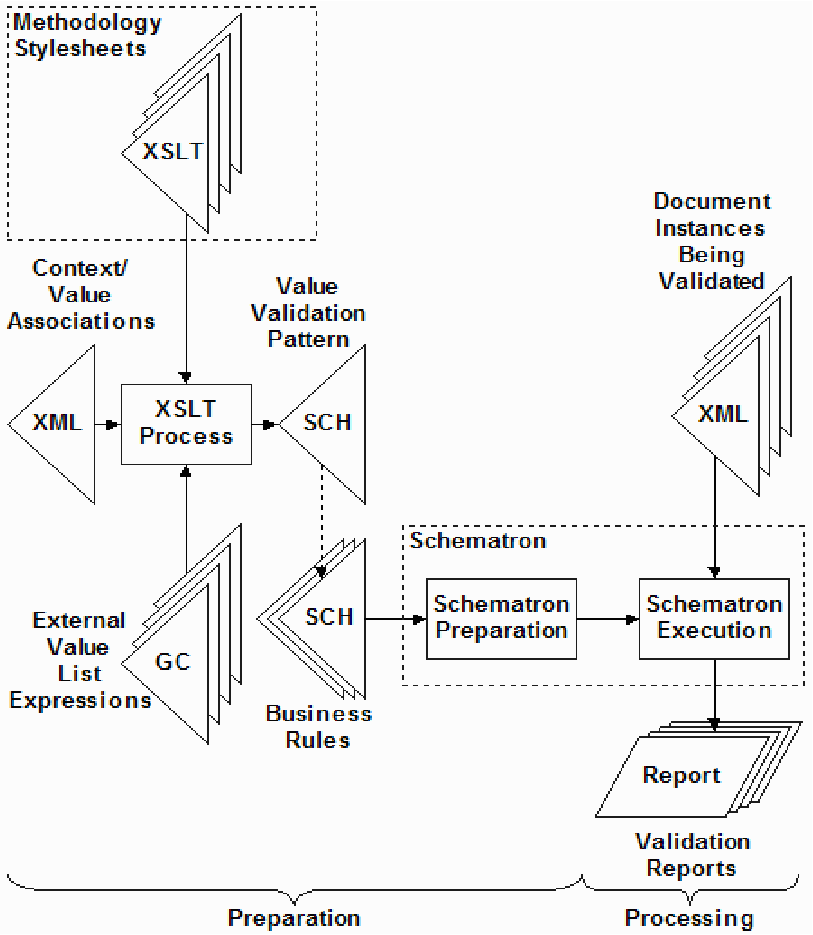
Schemata describe the structural and lexical constraints on a document. Some information items in a document are described in the schema lexically as a simple value whereby the value is a code representing an agreed-upon semantic concept. The value used is typically chosen from a set of unique coded values enumerating related concepts. This methodology is in support of the second pass of a two-pass validation strategy, where the "first pass" confirms the structural and lexical constraints of a document and the "second pass" confirms the value constraints of a document.

The "first pass" can be accomplished with an XML document schema language such as W3C Schema or ISO/IEC 19757-2 RELAXNG; "the second pass" is accomplished with a transformation language such as a W3C XSLT 1.0 stylesheet or a Python program. The second pass is as an implementation of ISO/IEC 19757-3 Schematron schemas that are utilized by this methodology.

In the figure below, "Methodology context association" depicts a file of context/value associations in the lower centre, where each association specifies for information items in the document instance being validated which lists of valid values in external value list expressions are to be used.


Document Instance Being Validated
External Value List Expressions
Lists of vaid values are referenced by file location
Document context specifies information items being validated
Context/ Value Associations
An association ties a document context to one or more lists of valid values

ISO Schematron is a powerful and yet simple assertion-based schema language used to confirm the success or failure of a set of assertions made about XML document instances. One can use ISO Schematron to express assertions supporting business rules and other limitations of XML information items so as to aggregate sets of requirements for the value validation of documents. The synthesis of a pattern of ISO Schematron assertions to validate the values found in document contexts, and the use of ISO Schematron to validate those assertions are illustrated in "Methodology overview" figure below.



To feed the ISO Schematron process, one needs to express the contexts of information items and the values used in those contexts. This methodology prescribes an XML vocabulary to create instances that express such associations of values for contexts. The stylesheets provided with this methodology read these instances of context/value associations that point to externally-expressed lists of values and produce an ISO Schematron *pattern* of assertions that can then be combined with other patterns for business rule assertions to aggregate all document value validation requirements into a single process. The validation process is then used against documents to be validated, producing for each document a report of that document's failures of assertions.

By using this methodology, users can use a default code list values for data exchange by adding more values to the default code list or restricting the values in the default code lists by defining constraints and business rules.

3.4.3.2 Representing Genericode based Code Lists (Option 2)

Each code list for a CIQ entity (Name, Address, and Party) is represented as a separate genericode file. For example, the Name entity has 10 code lists. Each of this code lists is represented in a separate genericode file.

3.4.3.2.1 Code List Representation in Genericode (Option 2) – An Example

The following example shows Genericode representation of code list for *SubDivisionTypeList* represented in a file called “SubDivisionTypeList.gc”.

```
<CodeList>
   <SimpleCodeList>
      <Row>
          <Value ColumnRef=”code”>
              <SimpleValue>Department</SimpleValue>
          </Value>
          <Value ColumnRef=”name”>
              <SimpleValue>Department</SimpleValue>
          </Value>
      </Row>
      <Row>
          <Value ColumnRef=”code”>
              <SimpleValue>Division</SimpleValue>
          </Value>
          <Value ColumnRef=”name”>
              <SimpleValue>Division</SimpleValue>
          </Value>
      </Row>

      <<Row>
          <Value ColumnRef=”code”>
              <SimpleValue>Branch</SimpleValue>
          </Value>
          <Value ColumnRef=”name”>
              <SimpleValue>Branch</SimpleValue>
          </Value>
      </Row>
      <Row>
          <Value ColumnRef=”code”>
              <SimpleValue>BusinessUnit</SimpleValue>
          </Value>
          <Value ColumnRef=”name”>
              <SimpleValue>BusinessUnit</SimpleValue>
          </Value>
      </Row>
      <Row>
          <Value ColumnRef=”code”>
              <SimpleValue>Section</SimpleValue>
          </Value>
          <Value ColumnRef=”name”>
              <SimpleValue>Section</SimpleValue>
          </Value>
      </Row>
   </SimpleCodeList>
</CodeList>
```

3.4.3.3 Customizing Genericode based Code Lists (Option 2)

Taking the same example of customizing code lists in Option 1, *OrganisationNameTypeList* code list will be a separate file called “*OrganisationNameTypeList.gc*”. To create a completely new set of code lists to replace the default one, a new .gc file with the new set of code list values say, “*ReplaceOrganisationNameTypeList.gc*” is created. By applying the constraints rule in a separate file, this new code list replaces the default code list.

The process of customizing the code lists is documented in the methodology for code list and value validation.

3.4.3.4 CIQ Specifications used as a case study by OASIS Code List TC

The OASIS Code List Technical Committee has used OASIS CIQ Specification V3.0’s Name entity (*xNL.xsd*) as a case study to demonstrate how genericode based code list approach can be used to replace XML schema approach to validate code lists (the default approach used by CIQ Specifications). This document is listed in the reference section.

3.4.3.5 References for Option 2

Following are the documents that users of CIQ Specifications implementing Genericode based Code List (Option 2) approach MUST read and understand:

- OASIS Codelist Representation (Genericode) Version 1.0, May 2007, http://docs.oasis-open.org/codelist/cd-genericode-1.0/doc/oasis-code-list-representation-genericode.pdf
- OASIS UBL Methodology for Codelist and value validation, Working Draft 0.8, November 2006, http://www.oasis-open.org/committees/document.php?document_id=21324
- OASIS Code List Adaptation Case Study (OASIS CIQ), May 2007, http://www.oasis-open.org/committees/document.php?document_id=23711

3.5 Code List Packages – Option 1 and Option 2

CIQ Specification comes with two sets of supporting CIQ entity XML schema packages, one for option 1 and the other for option 2 of code lists. To assist users in getting a quick understanding of option 2, all code lists for CIQ specifications are represented as genericode files along with default constraints, appropriate XSLT to process code lists, and with sample test XML document instance examples. It also contains test scenarios with customised code lists from the default code lists along with business rules, constraints supporting the customised code lists, XSLT and sample XML document instance examples.

The CIQ Specification entity schemas (*xNL.xsd, xAL.xsd, xPIL.xsd*, and *xNAL.xsd*) for both option 1 and 2 are in the same namespaces as users will use one of the two. XML document instances of Option 1 can be validated against the entity schemas in option 2. This is not true vice versa as Option 2 entity schemas have additional metadata attributes to support genericode.

A separate document titled, “CIQ Specifications V3.0 Package” explains the structure of the CIQ Specifications V3.0 package.

3.6 Order of Elements and Presentation

Order of name elements MUST be preserved for correct presentation (e.g. printing name elements on an 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.

3.6.1 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.

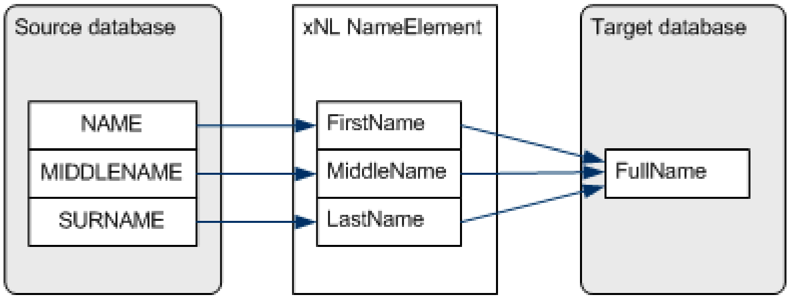
3.7 Data Mapping

Mapping data between the *xNL* schema and a target database is not expected to be problematic 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.*

For example, consider a data provider that has a person name in one line (free text and unparsed) 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 smart name parsing data quality parsing/scrubbing tool and there are plenty in the market. Such parsing/scrubbing is expected to be the responsibility of the data consumer under this scenario and importantly, agreeing in advance with the data provider that the incoming data is not parsed.

3.7.1 Example – Complex-to-simple Mapping

The source database easily maps to the *xNL NameElement* qualified with the *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>
```

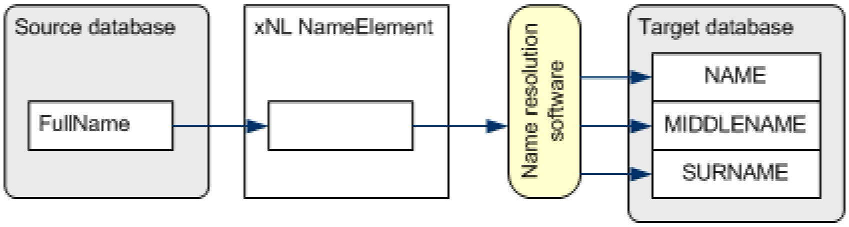
Target Database

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.

3.7.2 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

3.8 Data Quality

The quality of any information management/processing system is only as good as the quality of the data it processes/stores/manages. No matter how efficient the interoperability of data is, if the quality of data that is interoperated is poor, the business benefit arising out of the information processing system is expected to be poor. To structurally represent the data, understand the semantics of the data to integrate and interoperate the data, quality of the data is critical. CIQ specifications have been designed with the above formula in mind.

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*" (default values), if such status is known. If *DataQuality* attribute is omitted, it is presumed that the validity of the data is unknown. Users can customize the DataQuality code list to add more data quality attributes if required.

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

3.8.1 Example – Data Quality

```
<n:PersonName n:DataQuality="Valid"
             n: ValidFrom="2001-01-01T00:00:00"
   <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 and the validity of the data has been recorded as of 2001-01-01.

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.

3.8.2 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.

3.8.3 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.

3.9 Extensibility

All elements in *Name*, *Address* and *Party* namespaces support extensibility by allowing for any number of attributes from a non-target namespace to be added. This is allowed in the XML Schema specifications of CIQ.

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. Extensions without agreements between data exchange parties will break interoperability.

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.

3.9.1 Extending the Schemas to Meet Application Specific Requirements

CIQ Specifications does its best to provide the minimum required set of elements and attributes that are commonly used independent of applications to define party data (name, address and other party attributes). If specific applications require some additional set of attributes that are not defined in CIQ specifications, then this extensibility mechanism SHOULD be used provided the extensions are agreed with other parties in case of data exchange involving more than one application. If no agreement is in place, interoperability will not be achieved. Use of this extensibility mechanism SHOULD be governed.

3.9.2 Practical Applications

3.9.2.1 System-specific Identifiers

Participants involved in data exchanges 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.

3.9.2.2 Additional Metadata

Sometimes it MAY be 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>
```

In the above examples, “OperatorID” and “Corrected” are additional metadata added to “PartyName” from different namespaces without breaking the structure of the schema.

3.10 Linking and Referencing

Linking and referencing of different resources such as Party Name or Party Address (internal to the document or external to the document) can be achieved by two ways. It is important for parties involved in data exchange SHOULD decide during design time the approach they will be implementing. Implementing both the options will lead to interoperability problems. Just choose one. The two options are:

- Using *xLink*
- Using Key Reference

3.10.1 Using xLink [OPTIONAL]

CIQ has now included support for *xLink* style referencing. These attributes are OPTIONAL and so will not impact implementers who want to ignore them. The *xLink* attributes have been associated with extensible type entities within the CIQ data structure thereby allowing these to be externally referenced to support dynamic value lists. The *xBRL* (extensible Business Reporting Language) standards community for example, uses this approach extensively to indicate the type values of objects in the data structure.

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 in the CIQ specifications MAY slightly differ from the original *xLink* specification. See *CIQ TC Party Relationships Specification* for more information on using *xLink* with *xNL* [Not available in this version]. 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*.

Implementers are not restricted to only using *XLink* for this purpose - for example the xlink:href attribute MAY be re-used for a URL to a REST-based lookup, and so forth. The intent is to provide additional flexibility for communities of practice to develop their own guidelines when adopting CIQ.

3.10.2 Using Key Reference [OPTIONAL]

This approach MAY be used for internal references (i.e. within some XML infoset that contains both referencing and referenced elements).

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

```
<c:Customers
   xmlns:c="urn:acme.org:corporate:customers"
   xmlns:a="urn:oasis:names:tc:ciq:xal:3"
   xmlns:n="urn:oasis:names:tc:ciq:xnl:3"
   xmlns:p="urn:oasis:names:tc:ciq:xpil:3"
   <p:Party PartyKey=”111”>
     <n:PartyName>
       <n:PersonName>
           <n:NameElement n:ElementType=”FirstName”>Ram</n:NameElement>
           <n:NameElement n:ElementType=”LastName”>Kumar</n:LastName>
       </n:PersonName>
     </n:PartyName>
     <p:Party p:PartyKey=”222”>
       <n:PartyName>
          <n:PersonName>
             <n:NameElement n:ElementType=”FirstName”>Joe</n:NameElement>
             <n:NameElement n:ElementType=”LastName”>Sullivan</n:LastName>
          </n:PersonName>
       </n:PartyName>
     </p:Party>
     <c:Contacts>
           <c:Contact c:PartyKeyRef=”222”>
           <c:Contact c:PartyKeyRef=”111”>
     <c:/Contacts>
</c:Customers>
```

3.11 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 MUST be unique within the scope of the application of xNL and the value MUST be globally unique. The term ‘globally unique’ means a unique identifier that is “mathematically guaranteed” to be unique. For example, GUID (Globally 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 need 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.

3.12 Schema Conformance

Any XML documents produced MUST conform to the CIQ Specifications Schemas namely, *xNL.xsd, xAL.xsd, xNAL.xsd* and *xPIL.xsd* i.e. the documents MUST be successfully validated against the Schemas. This assumes that the base schemas MUST be modified.

If Option 2 for Code List is used, all genericode files MUST conform to the Genericode XML Schema, i.e. all genericode files MUST successfully validate against the schema.

Any customisation of the code list files based on Option 1 MUST be well formed schemas.

3.13 Schema Customization Guidelines

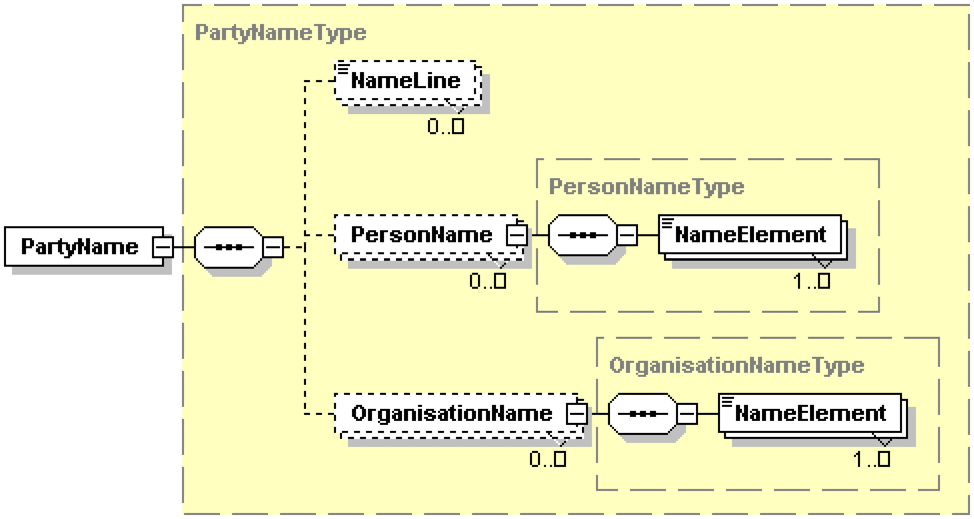
The broad nature and cultural diversity of entity "Name" makes it very difficult to produce one schema that would satisfy all applications and all cultures while keeping the size and complexity of the schema manageable. This specification allows some changes to the schema by adopters of the schema to fit their specific requirements and constraints. However, note that any change to the schema breaks the CIQ Specifications compatibility and so, they MUST NOT be changed.

3.13.1 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.

3.13.2 Reducing the Entity Schema Structure

Users SHOULD retain the minimum structure of Name entity as in the following diagram:



This structure allows for most names to be represented, with exception for

- 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.

Users MUST NOT remove any attributes from the schema. Attributes in the schema can be easily ignored during the processing.

3.13.2.1 Implications of changing Name Entity Schema

Any changes to the Name Entity schema (*xNL.xsd*) are likely to break the compatibility one way or another.

It MAY be possible that an XML fragment created for the original schema is invalid for the altered schema or vice versa. This issue SHOULD be considered before making any changes to the schema that could break the compatibility.

3.13.3 Customizing the Code Lists/Enumerations of Name

Meeting all requirements of different cultures and ethnicity in terms of representing the names in one specification is not trivial. This is the reason why code lists/enumerations are introduced in order to keep the specification/schema simple, but at the same time provide the flexibility to adapt to different requirements.

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

> **NOTE:** The code list/enumeration values for different enumeration lists that are provided as part of the specification are not complete. They only provides some sample values and it is up to the end users to customise them to meet their data exchange requirements if the default values are incomplete, not appropriate or over kill

This level of flexibility allows some customization of the schema through changing the code list/enumerations only, without changing the basic structure of the schema. It is important to ensure that all schema users involved in data exchange use the same code list/enumerations for interoperability to be successful. This has to be negotiated between the data exchange parties and a proper governance process SHOULD be in place to manage this process.

3.13.4 Using the Code list Methodology (UMCLVV) to customize Name Schema to meet application specific requirements

The other approach to customize the CIQ Name schema (includes other entity schemas namely Party and Address) without touching it is by using the UML Methodology for Code List Value and Validation (UMCLVV). In this approach, one can use Schematron patterns to define assertion rules to customize the *xNL.xsd* schema without modifying it. For example, it is possible to customize *xNL.xsd* schema to restrict the use of elements, the occurrence of elements, the use of attributes, and the occurrence of attributes, making elements and attributes mandatory, etc.

So, users who believe that many elements and attributes in the CIQ specifications are overwhelming to what their requirements are, can define business rules using Schematron patterns to constraint the CIQ base entity schemas. By constraining the CIQ schemas, users get two major benefits:

- CIQ Specifications that are tailored indirectly with the help of business rules to meet specific application requirements
- Applications that use the customized CIQ Specifications with the help of business rules are still compliant with the CIQ Specifications.

Therefore, by CIQ specifications providing two options for customizing schemas (Option 1 and Option 2), the specifications are powerful to address any specific application requirements for party information.

> **NOTE:** The business rules used to constraint base schemas SHOULD be agreed by all the parties that are involved in CIQ based data exchange to ensure interoperability and the rules SHOULD be governed.

3.13.4.1 Constraining Name Schema using UMCLVV – An Example

xNL.xsd uses “NameElement” element as part of “PersonName” element to represent the name of a person and this is supported by using the attribute “ElementType” to add semantics to the name. Let us look at the following example:

```
<n:PersonName>
   <n:NameElement n:ElementType=”FirstName>Paruvachi</n:NameElement>
   <n:NameElement n:ElementType=”FirstName>Ram</n:NameElement>
   <n:NameElement n:ElementType=”MiddleName>Kumar</n:NameElement>
   <n:NameElement n:ElementType=”LastName>Venkatachalam</n:NameElement>
   <n:NameElement n:ElementType=”LastName>Gounder</n:NameElement>
</n:PersonName>
```

In the above example, there is no restriction on the number of times *First Name* and *Last Name* can occur as per *xNL.xsd* schema grammar. Some applications might want to apply strict validation and constraint rules on the *xNL.xsd* schema to avoid use of *First Name* and *Last Name* values to data at least once and no more than once. This is where UMCLVV can be used to define business rules to constraint the *xNL.xsd* schema without modifying or touching the *xNL.xsd* schema. The business rule code defined using Schematron pattern for the above constraint is given below:

```
<rule context=”n:PersonName[not(parent::n:PartyName)]”>
   <assert test=count(n:NameElement [@n:ElementType=’FirstName’]=1”
          >Must have exactly one FirstName component</assert>
   <assert test=count(n:NameElement[@n:ElementType=’LastName’])=1”
          >Must have exactly one LastName component</assert>
</rule>
```

When first pass validation (structure validation) is performed on the above sample XML instance document, the document is valid against the *xNL.xsd.* During second pass validation (business rule constraint and value validation) on the above XML instance document, the following error is reported:

```
Must have exactly one FirstName component
Must have exactly one LastName component
```

4 Entity “Address” (extensible Address Language)

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 2.4 Common Design Concepts for more information.

4.1 Semantics of “Address”

The high level schema elements of *xAL* schema are illustrated in the diagram in the next page.

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

4.1.1 Example – Minimal Semantics (Unstructured/Free Text Data)

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. Data represented in this free formatted text form is classified as “poor quality” as it is subject to different interpretations of the data and will cause interoperability problems.

Many common applications fall under this category.

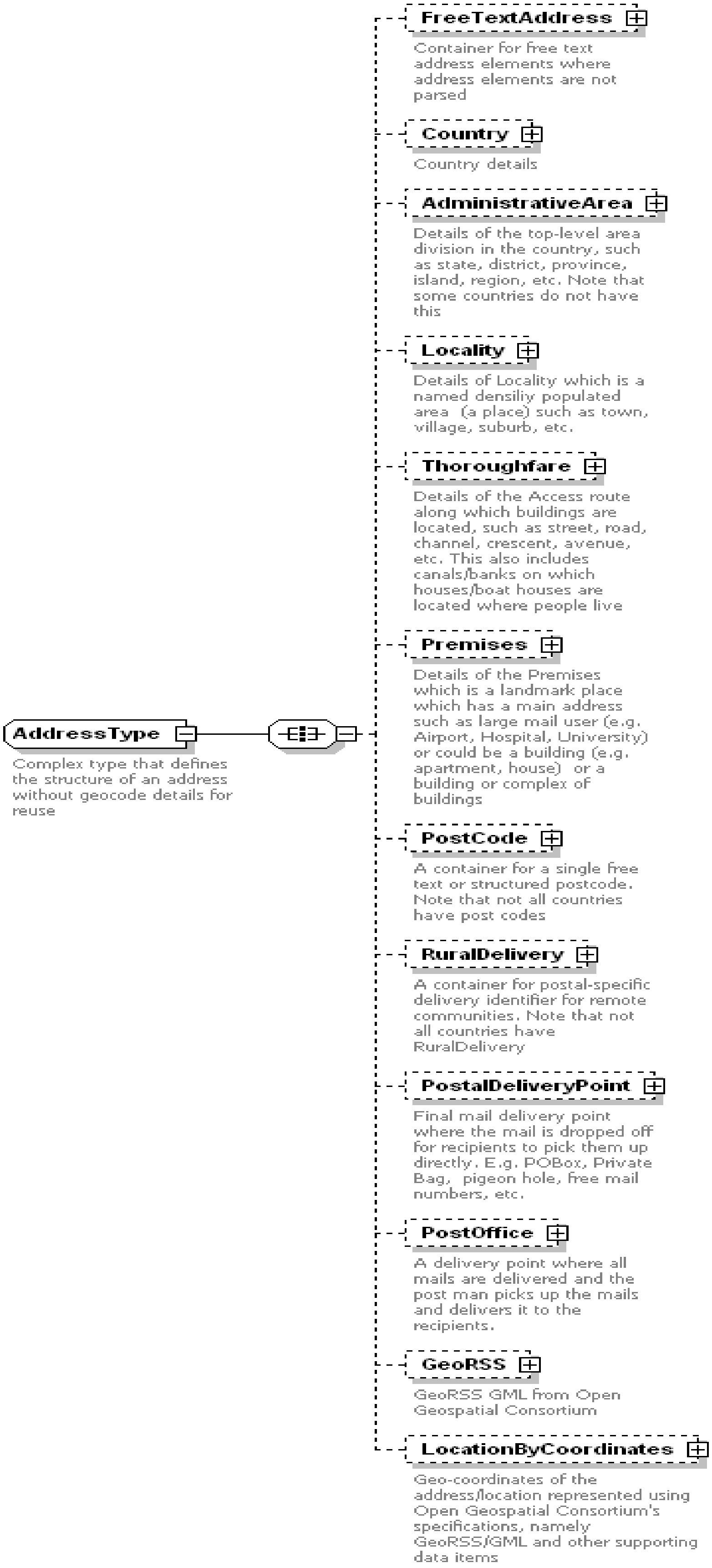
4.1.2 Example – Partial Semantics (Partially Structured Data)

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.

Many common applications fall under this category.


AddressType
Complex type that defines the structure of an address without geocode details for reuse
FreeTextAddress
Container for free text address elements where address elements are not parsed
Country
Country details
AdministrativeArea
Details of the top-level area division in the country, such as state, district, province, island, region, etc. Note that some countries do not have this
Locality
Details of Locality which is a named densiliy populated area (a place) such as town, village, suburb, etc.
Thoroughfare
Details of the Access route along which buildings are located, such as street, road, channel, crescent, avenue, etc. This also includes canals/banks on which houses/boat houses are located where people live
Premises
Details of the Premises which is a landmark place which has a main address such as large mail user (e.g. Airport, Hospital, University) or could be a building (e.g. apartment, house) or a building or complex of buildings
PostCode
A container for a single free text or structured postcode. Note that not all countries have post codes
RuralDelivery
A container for postal-specific delivery identifier for remote communities. Note that not all countries have RuralDelivery
PostalDeliveryPoint
Final mail delivery point where the mail is dropped off for recipients to pick them up directly. E.g. POBox, Private Bag, pigeon hole, free mail numbers, etc.
PostOffice
A delivery point where all mails are delivered and the post man picks up the mails and delivers it to the recipients.
GeoRSS
GeoRSS GML from Open Geospatial Consortium
LocationByCoordinates
Geo-coordinates of the address/location represented using Open Geospatial Consortium's specifications, namely GeoRSS/GML and other supporting data items

4.1.3 Example – Full Semantics (Fully Structured Data)

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

```
<a:Address>
   <a:AdministrativeArea a:Type="state">
          <a:NameElement a:Abbreviation="true" a:NameType="Name">VIC</a:NameElement>
   </a:AdministrativeArea>
   <a:Locality a:Type="suburb">
          <a:NameElement a:NameType="Name">CLAYTON</a:NameElement>
          <a:SubLocality a:Type="Area">
              <a:NameElement a:NameType="Name">Technology Park</a:NameElement>
           </a:SubLocality>
   </a:Locality>
   <a:Thoroughfare a:Type="ROAD">
          <a:NameElement a:NameType="NameandType">Dandenong Road</a:NameElement>
          <a:Number a:IdentifierType="RangeFrom">200</a:Number>
          <a:Number a:IdentifierType="Separator">-</a:Number>
          <a:Number a:IdentifierType="RangeTo">350</a:Number>
          <a:SubThoroughfare a:Type=”AVENUE”>
                  <a:NameElement a:NameType="NameAndType">Fifth Avenue</a:NameElement>
          </a:SubThoroughfare>
   </a:Thoroughfare>
   <a:Premises a:Type="Building">
          <a:NameElement a:NameType="Name">Toshiba Building</a:NameElement>
   </a:Premises>
   <a:PostCode>
          <a:Identifier>3168</a:Identifier>
   </a:PostalCode>
</a:Address>
```

Few applications and in particular, applications dealing with data quality and integrity, fall in this category and the quality of data processed by these applications are generally high.

4.2 Address/Location Referenced By GeoRSS and Coordinates

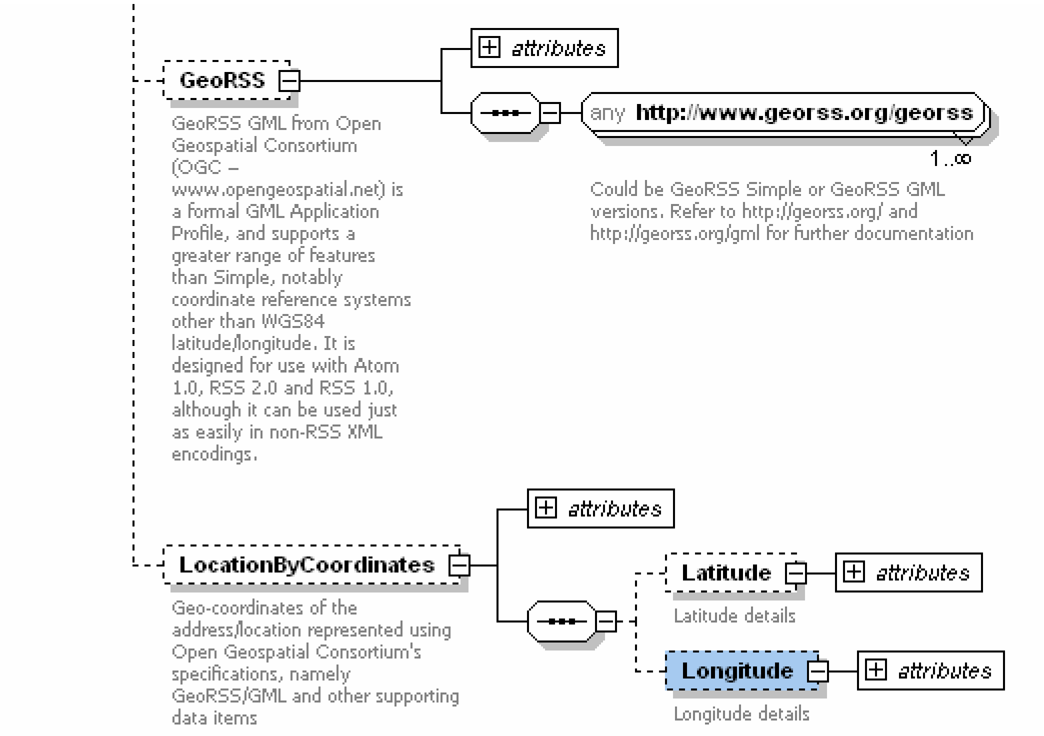
xAL supports representation of Address/location in two ways namely,

1. By using explicit coordinates with qualifiers for accuracy and precision, and
2. By using the GeoRSS application profile, which expresses decimal degrees coordinates with accuracy and precision, and is implemented via external namespaces (either ATOM or RSS).

Explicit coordinates are typically available from the process of geo-coding the street addresses. Coordinates are expressed in the *Latitude* and *Longitude* elements, including *DegreesMeasure, MinutesMeasure, SecondsMeasure,* and *Direction.* Data quality is expressed as attributes of coordinates including *Meridian, Datum and Projection.*

GeoRSS 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 basic expression of *a:LocationByCoordinate* element in *xAL.xsd* schema has limits in utility for e-commerce applications. More interoperable expression of coordinate is possible via GeoRSS, due to the ability to reduce ambiguity introduced by requirements for different coordinate systems, units and measurements, or the ability to define more complex (non-point) geographic features.

Support for GeoRSS and Location Coordinates for address/locations in *xAL.xsd* schema is shown in the following figure.


GeoRSS
attributes
any http://www.georss.org/georss
1..∞
GeoRSS GML from Open Geospatial Consortium (OGC – www.opengeospatial.net) is a formal GML Application Profile, and supports a greater range of features than Simple, notably coordinate reference systems other than WGS84 latitude/longitude. It is designed for use with Atom 1.0, RSS 2.0 and RSS 1.0, although it can be used just as easily in non-RSS XML encodings.
Could be GeoRSS Simple or GeoRSS GML versions. Refer to http://georss.org/ and http://georss.org/gml for further documentation
attributes
LocationByCoordinates
Geo-coordinates of the address/location represented using Open Geospatial Consortium's specifications, namely GeoRSS/GML and other supporting data items
Latitude
attributes
Latitude details
Longitude
attributes
Longitude details

4.2.1 Using GeoRSS in xAL Schema

As RSS becomes more and more prevalent as a way to publish and share information, it becomes increasingly important that location is described in an interoperable manner so that applications can **request**, **aggregate**, **share** and **map** geographically tagged feeds.

GeoRSS (Geographically Encoded Objects for RSS feeds) enables geo-enabling, or tagging, "really simple syndication" (RSS) feeds with location information. GeoRSS proposes a standardized way in which location is encoded with enough simplicity and descriptive power to satisfy most needs to describe the location of Web content. GeoRSS MAY not work for every use, but it should serve as an easy-to-use geo-tagging encoding that is brief and simple with useful defaults but extensible and upwardly-compatible with more sophisticated encoding standards such as the OGC (Open Geospatial Consortium) GML (Geography Markup Language).

GeoRSS was developed as a collaborative effort of numerous individuals with expertise in geospatial interoperability, RSS, and standards, including participants in the -- the W3C (World Wide Web Consortium)[1] and OGC (Open Geospatial Consortium)[2].

GeoRSS is a formal GML Application Profile, with two flavours: 'GeoRSS Simple', which describes a point, and 'GeoRSS GML', which describes four essential types of shapes for geo-referencing (point, line, box and polygon).

[1] OGC – www.opengeospatial.net

[2] W3C – www.w3c.org

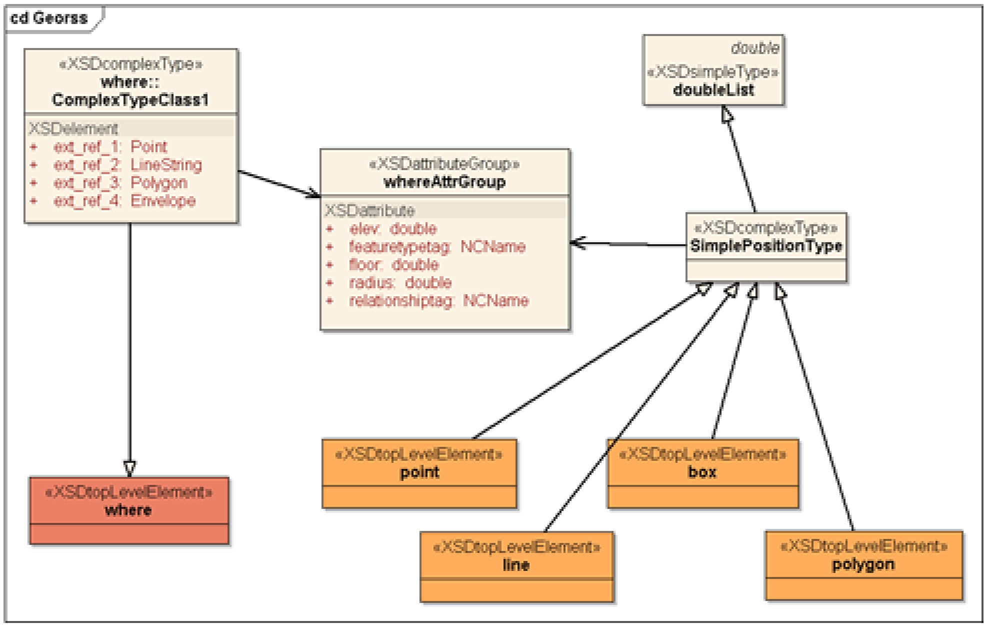
GeoRSS Simple has greater brevity, but also has limited extensibility. When describing a point or coordinate, GeoRSS Simple can be used in all the same ways and places as GeoRSS GML.

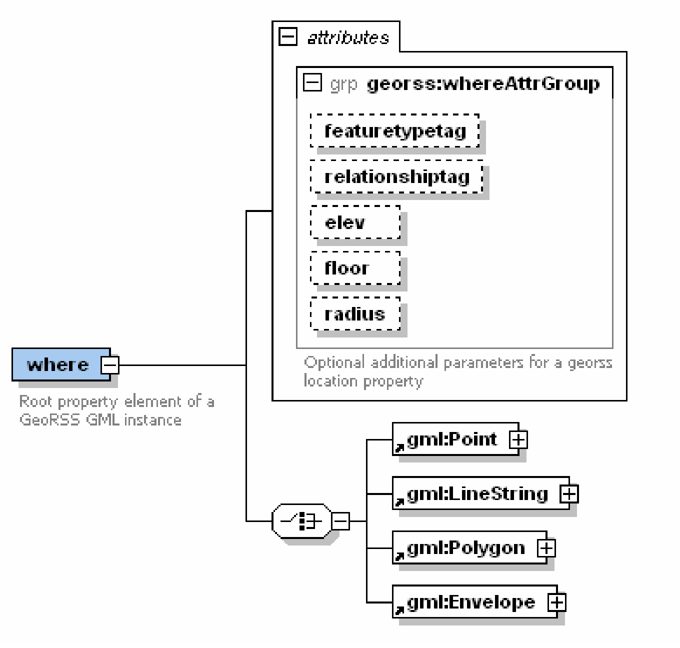
GeoRSS GML supports a greater range of features, notably coordinate reference systems other than WGS84 latitude/longitude. It is designed for use with Atom 1.0, RSS 2.0 and RSS 1.0, although it can be used just as easily in non-RSS XML encodings.

Further detailed documentation and sample xml implementation information are published on the sites listed below:

- http://georss.org/
- http://georss.org/gml
- http://georss.org/atom

The UML model for the GeoRSS application schema and the XML schema is shown below:




attributes
grp georss:whereAttrGroup
featuretypetag
relationshiptag
elev
floor
radius
Optional additional parameters for a georss location property
where
Root property element of a GeoRSS GML instance
gml:Point
gml:LineString
gml:Polygon
gml:Envelope

GeoRSS is supported by an element *a:GeoRSS* in *xAL.xsd* schema as a non target namespace. The content of *a:GeoRSS* must comply with the following requirements:

- Be from the GeoRSS/GML/Atom namespace
- Refer to finest level of address details available in the address structure that a:GeoRSS 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:GeoRSS* can describe be it a point, linear feature, polygon or a rectangle.

4.2.1.1 GeoRSS - Example

The following are GeoRSS examples and demonstrate what GeoRSS Simple and GeoRSS GML encodings look like. The location being specified is city center Ft. Collins.

Simple GeoRSS:

```
<georss:point>40.533203 -105.0712</georss:point>
```

GML GeoRSS:

```
<GeoRSS:where>
   <gml:Point>
      <gml:pos>40.533203 -105.0712</gml:pos>
   </gml:Point>
<GeoRSS:where>
```

These examples are in XML. However, RSS and GeoRSS are general models that can also be expressed in other serializations such as Java, RDF or XHTML.

4.2.1.2 GeoRSS GML – Example

A good way to describe a trip that has many places of interest like a boat trip or a hike is to specify the overall trip's path with a line as a child of the <feed>. Then mark each location of interest with a point in the <entry>.

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:georss="http://www.georss.org/georss"
      xmlns:gml="http://www.opengis.net/gml">
   <title>Dino's Mt. Washington trip</title>
   <link href="http://www.myisp.com/dbv/"/>
   <updated>2005-12-13T18:30:02Z</updated>

   <author>
      <name>Dino Bravo</name>
      <email>dbv@example.org</email>
   </author>

   <id>http://www.myisp.com/dbv/</id>

   <georss:where>
      <gml:LineString>
         <gml:posList>
            45.256 -110.45 46.46 -109.48 43.84 -109.86 45.8 -109.2
         </gml:posList>
      </gml:LineString>
   </georss:where>

   <entry>
      <title>Setting off</title>
      <link href="http://www.myisp.com/dbv/1"/>
      <id>http://www.myisp.com/dbv/1</id>
      <updated>2005-08-17T07:02:32Z</updated>
      <content>getting ready to take the mountain!</content>
      <georss:where>
         <gml:Point>
            <gml:pos>45.256 -110.45</gml:pos>
         </gml:Point>
      </georss:where>
   </entry>

   <entry>
      <title>Crossing Muddy Creek</title>
      <link href="http://www.myisp.com/dbv/2"/>
      <id>http://www.myisp.com/dbv/2</id>
      <updated>2005-08-15T07:02:32Z</updated>
      <content>Check out the salamanders here</content>
      <georss:where>
         <gml:Point>
            <gml:pos>45.94 -74.377</gml:pos>
         </gml:Point>
      </georss:where>
   </entry>
</feed>
```

4.2.2 Defining Location Coordinates in xAL Schema

If end users feel that GeoRSS GML is “overkill” or complex for their requirement and instead, want to just define the coordinates for location/address, *xAL.xsd* schema provides a default set of basic and commonly used elements representing explicit location coordinates through the element *a:LocationByCoordinates.*

a:LocationByCoordinates element provides attributes namely, *Datum*, type of code used for Datum, *Meridian*, type of code used for Meridian, *Projection* and type of code used for Projection.

a:LocationByCoordinates/a:Latitude and *a:LocationByCoordinates/a:Longitude* elements provide attributes namely, *DegreesMeasure, MinutesMeasure, SecondsMeasure, and Direction.*

4.3 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 a simple type “*String*” (255 characters in size and collapsed white spaces) defined in *CommonTypes.xsd*. Other XML Schema data types are also used throughout the schema.

Other XML Schema defined data types (e.g. int, string, DateTime) are also used throughout xAL namespace.

4.4 Code Lists (Enumerations)

Use of code lists/enumerations is identical to use of code lists/enumerations for entity “*Name*”. Refer to section 3.3 for more information.

Code Lists used in *xAL* for Option 1 reside in an “include” file *xAL-types.xsd* and for option 2 as separate genericode files.

NOTE: The code list values for different code lists that are provided as part of the specifications are not complete. They only provides some sample values and it is up to the end users to customise them to meet their data exchange requirements if the default values are incomplete, not appropriate or an over kill

4.5 Order of Elements and Presentation

Order of address elements MUST be preserved for correct presentation in a fashion similar to what is described in section 3.6.

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

4.5.1 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>
```

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

4.6 Data Mapping

Mapping data between *xAL* schema and a database is similar to that of entity “*Name*” as described in section 3.7.

4.6.1 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.

4.7 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 3.8 for more information.

4.8 Extensibility

All elements in *Address* namespace are extensible as described in section 3.9.

4.9 Linking and Referencing

All linking and referencing rules described in section 3.10 apply to entity "*Address*".

Use of attribute ID is described in section 3.11.

4.10 Schema Conformance

Schema conformance described in section 3.12 is fully applicable to entity "*Address*".

4.11 Schema Customization Guidelines

Schema customisation rules and concepts described in section 3.13 are fully applicable to entity "*Address*".

4.11.1 Customizing the Code Lists/Enumerations of Address

Meeting the 240+ country address semantics in one schema and at the same time keeping the schema simple is not trivial. Some countries have a city and some do not, some countries have counties, provinces or villages and some do not, some countries use canal names to represent the property on the banks of the canal, and, some countries have postal codes and some do not.

Key components of international addresses that vary from country to country are represented in the specification using the schema elements namely, *Administrative Area*, *Sub Administrative Area*, *Locality*, *Sub Locality*, *Premises*, *Sub Premises*, *Thoroughfare*, and *Postal Delivery Point*. CIQ TC chose these names because they are independent of any country specific semantic terms such as City, Town, State, Street, etc. Providing valid and meaningful list of code lists/enumerations as default values to these elements that covers all countries is not a trivial exercise. These elements are therefore, customisable using code lists/enumerations to preserve the address semantics of each country which assists in improving the semantic quality of the address. To enable end users to preserve the meaning of the address semantics, the specification provides the ability to customise the schema using code lists/enumerations without changing the structure of the schema itself. At the same time, the schema structure remains intact.

For example, "State" defined in the code list/enumeration list for Administrative Area type could be valid for countries like India, Malaysia and Australia, but not for Singapore as it does not have the concept of "State". A value "Nagar" in the code list/enumeration list for Sub Locality type could be only valid for countries like India and Pakistan.

If there is no intent to use the code list/enumeration list for the above schema elements, the code list/enumeration list can be ignored. There is no absolute must rule that the default values for the enumeration lists provided by the specification must exist. The list can be empty also. As long as the code list/enumeration list values are agreed between the parties involved in data exchange (whether data exchange between internal business system or with external systems), interoperability is not an issue.

In Option 1 of representing code lists, the values clarifying the meaning of geographical entity types (e.g. *AdministrativeAreaType, LocalityAreaType*) in *xAL.xsd* were intentionally taken out of the main schema file into an "include" file (*xAL-types.xsd*) to make customisation easier. In Option 2 of Code List representation, these code lists are represented as separate .gc file in genericode format.

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

NOTE: The code lists values for different enumeration lists that are provided as part of the specification are not complete. They only provides some sample values and it is up to the end users to customise them to meet their data exchange requirements if the default values are incomplete, not appropriate or over kill

4.11.1.1 End User Customised Code List - An Example

In the example below, we use the country, Singapore. The default values provided by CIQ Specification for *AdministrativeAreaType* enumeration are given below. The user might want to restrict the values to meet only the address requirements for Singapore. Singapore does not have any administrative areas as it does not have state, city, or districts or provinces. So, the user can customise the schema by making the *AdministrativeAreaType* enumeration as an empty list as shown in the table below.

Original xAL values for AdministrativeAreaType List	**Possible end user customised values**
City	
State	
Territory	
Province	

This level of flexibility allows some customization of the schema through changing the enumerations only, without changing the basic structure of the schema. It is important to ensure that all schema users involved in data exchange use the same enumerations for interoperability to be successful. This has to be negotiated between the data exchange parties and a proper governance process SHOULD be in place to manage this process.

4.11.1.2 Implications of changing Address Entity Schema

Any changes to the Address Entity schema (*xAL.xsd*) are likely to break the compatibility one way or another.

It MAY be possible that an XML fragment created for the original schema is invalid for the altered schema or vice versa. This issue needs to be considered before making any changes to the schema that could break the compatibility.

4.11.2 Using the Code list Methodology (UMCLVV) to customize CIQ Address Schema to meet application specific requirements

The other approach to customize the CIQ address schema (*xAL.xsd*) without modifying it is by using the UMCLVV. In this approach, one can use Schematron patterns to define assertion rules to customize CIQ address schema without modifying it. For example, it is possible to customize CIQ address schema to restrict the use of address entitities that are not required for a specific country. For example, a country like Singapore will not need address entities namely, *Administrative Area*, *Sub Administrative Area*, *Sub Locality, Rural Delivery* and *Post Office*. These entities can be restricted using Schematron based assertion rules. Some might want to just use free text address lines and a few of the address entities like locality and postcode. Schematron assertion rules help users to achieve this.

```
NOTE: The business rules used to constraint CIQ address schema SHOULD be agreed by
all the parties that are involved in data exchange of CIQ based address data to
ensure interoperability and the rules SHOULD be governed.
```

4.11.2.1 Constraining CIQ Address Schema using UMCLVV – Example 1

Let us use the country “Singapore” as an example again. Let us say that the country “Singapore” only requires the following address entities defined in *xAL.xsd* and does not need the rest of the entities defined in *xAL.xsd* as they are not applicable to the country:

- Country
- Locality
- Thoroughfare
- PostCode

This restriction can be achieved without modifying the *xAL.xsd* schema and by applying the following schematron pattern rules outside of the schema as follows:

```
<rule context=”a:Address/*”>
  <assert test=”(name()=’a:Country’) or (name()=’a:PostCode’) or
                (name()=’a.Thoroughfare’) or (name()=’a:Locality’)”
    >Invalid data element present in the document
   </assert>
</rule>
```

The above simple rule restricts the use of other elements and attributes in *xAL.xsd* when an XML instance document is produced and validated.

Now let us take the following XML instance document:

```
<a:Address>
  <a:Country>
        <a:NameElement>Singapore</a:NameElement>
  </a:Country>
  <a:AdministrativeArea>
        <a:NameElement></a:NameElement>
  </a:AdministrativeArea>
  <a:Locality>
        <a:NameElement>NUS Campus</a:NameElement>
  </a:Locality>
  <a:Thoroughfare>
        <a:NameElement>23 Woodside Road</a:NameElement>
  </a:Thoroughfare>
  <a:Premises>
        <a:NameElement></a:NameElement>
  </a:Premises>
  <a:PostCode>
        <a:Identifier>51120</a:Identifier>
  </a:PostCode>
</a:Address>

```

When the above document instance is validated using UMCLVV, pass one validation (structure validation against *xAL.xsd*) will be successful. Pass two validation (business rules and value validation) will report the following errors:

```
Invalid data element present in the document
         :/a:Address/a:AdministrativeArea
Invalid data element present in the document
         :/a:Address/a:Premises
```

4.11.2.2 Constraining CIQ Address Schema using UMCLVV – Example 2

Let us consider another example where an application requires using only the free text address lines in *xAL.xsd* and no other address entities.

This restriction can be achieved without modifying the *xAL.xsd* schema and by applying the following schematron pattern rules outside of the schema as follows:

```
<rule context=”a:Address/*”>
  <assert test=”name()=’a:FreeTextAddress’”
    >Invalid data element present in the document
   </assert>
</rule>
```

The above simple rule restricts the use of elements and attributes other than “*FreeTextAddress*” element in *xAL.xsd* when an XML instance document is produced and validated.

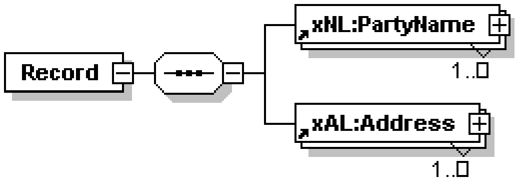
5 Combination of "Name" and "Address" (extensible Name and Address Language)

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

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

5.1 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 a person defined in the *xNL* part have some connection/link with an address specified in the *xAL* part. Use attributes from other namespace to specify the type of relationships and roles of names and addresses.

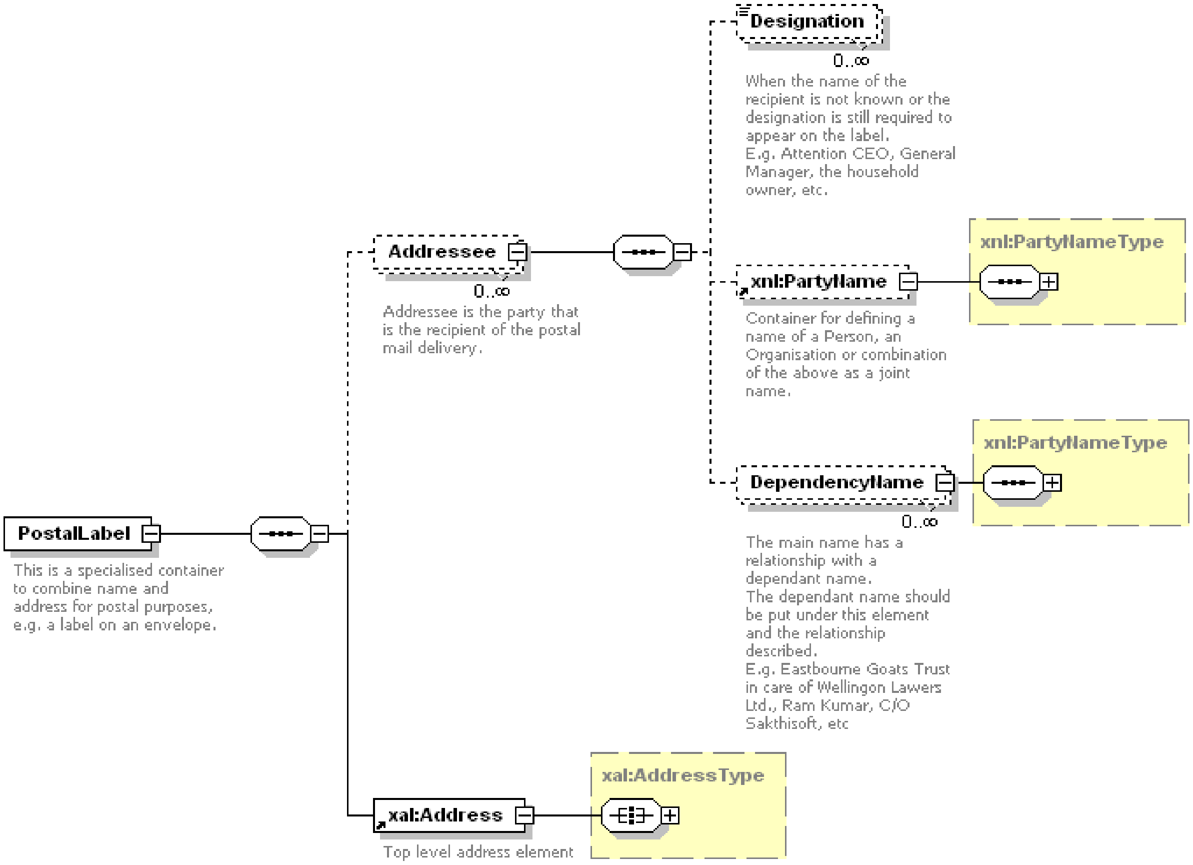
5.1.1 Example

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>
```

5.2 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 has two main containers, an addressee and the address:


Designation
0..∞
When the name of the recipient is not known or the designation is still required to appear on the label.
E.g. Attention CEO, General Manager, the household owner, etc.
Addressee
0..∞
Addressee is the party that is the recipient of the postal mail delivery.
xnl:PartyNameType
xnl:PartyName
Container for defining a name of a Person, an Organisation or combination of the above as a joint name.
xnl:PartyNameType
DependencyName
0..∞
The main name has a relationship with a dependant name.
The dependant name should be put under this element and the relationship described.
E.g. Eastbourne Goats Trust in care of Wellingon Lawers Ltd., Ram Kumar, C/O Sakthisoft, etc
PostalLabel
This is a specialised container to combine name and address for postal purposes, e.g. a label on an envelope.
xal:AddressType
xal:Address
Top level address element

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*.

5.2.1 Example

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

translates into the following *xNAL* fragment:

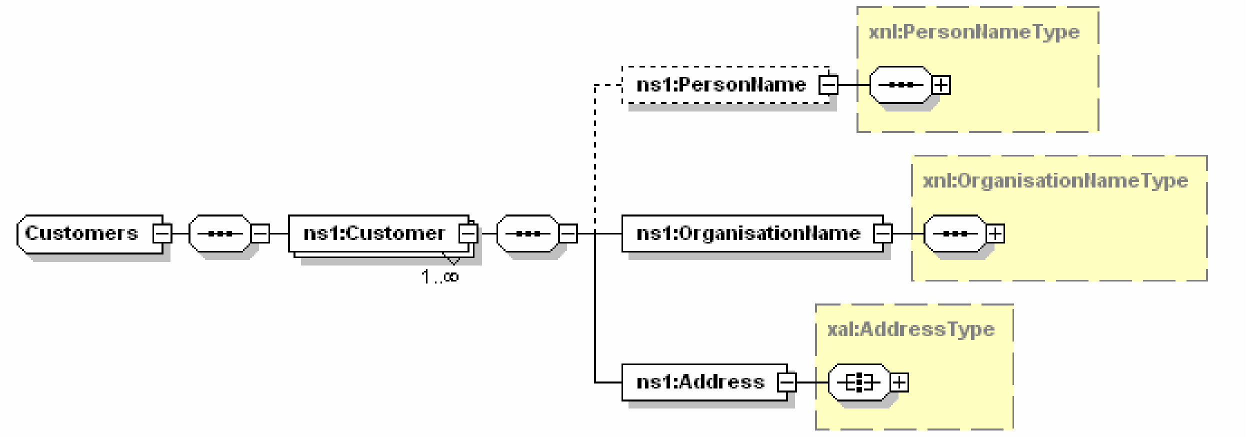
```
<xnal:PostalLabel>
   <xnal:Addressee>
          <xnal:Designation>Attention: Mr S Mart</xnal:Designation>
          <xnal:Designation>Director</xnal:Designation>
          <n:PartyName>
                 <n:NameLine>Name Plate Engravers</n:NameLine>
          </n:PartyName>
   </xnal:Addressee>
   <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>
```

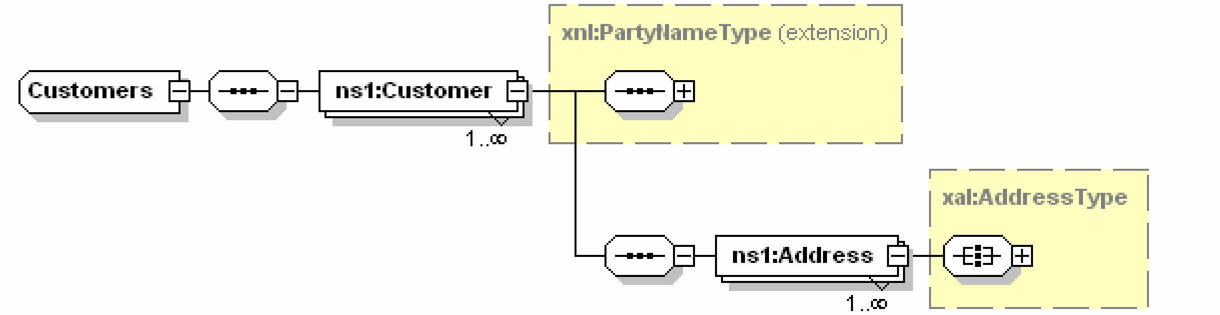
5.3 Creating your own Name and Address Application Schema

Users can use the *xNL* and *xAL* constructs and create their own name and address container schema to meet their specific requirements rather than using a container element called "Record" as in *xNAL* if they believe that *xNAL* schema does not meet their requirements. This is where the power of CIQ Specifications comes in to play. It provides the basic party constructs to enable users to reuse the base constructs of CIQ specifications as part of their application specific data model and at the same time meeting their application specific requirements.

For example, users can create a schema called *Customers.xsd* that could reuse *xNL* and *xAL* to represent their customers. This is shown in the following figure:



In the above figure, *PersonName* is OPTIONAL.



In the above figure, "Customer" is of type "Party" as defined in *xNL* schema. "Customer" is then extended to include "Address" element that is of type "Address" as defined in *xAL* schema.

6 Entity “Party” (extensible Party Information Language)

Entity “Party” encapsulates some most commonly used unique characteristics/attributes of *Person* or *Organisation*, such as name, address, personal details, contact details, physical features, etc.

This assists in uniquely identifying a party with these unique party attributes.

The schema consists of top level containers that MAY appear in any order or MAY be omitted. The containers are declared globally and can be reused by other schemas. The full schema for defining a *Party* can be found in *xPIL,xsd* file with enumerations in *xPIL-types.xsd* file for Code List Option 1 and .gc files for Code List Option 2. See the sample XML files for examples.

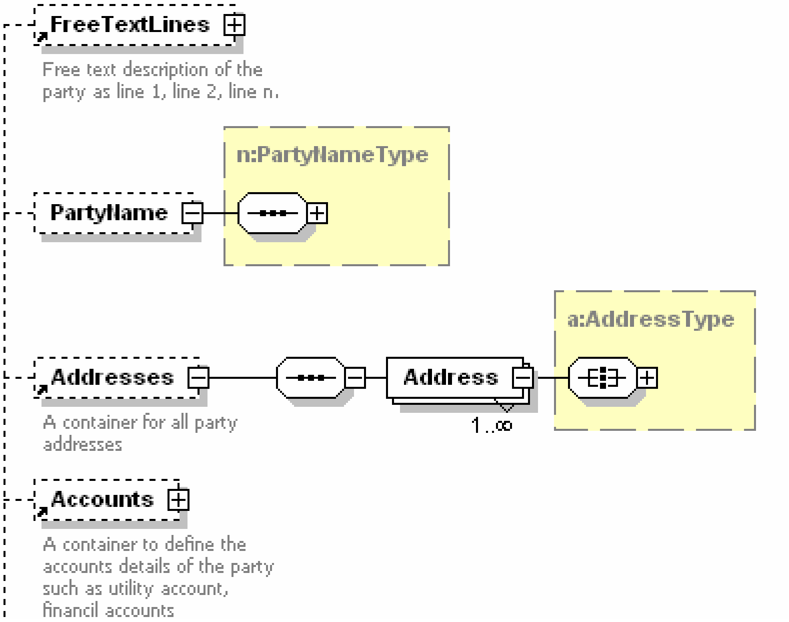
xPIL provides a number of elements/attributes that are common to both a person and an organisation (e.g. account, electronic address identifier, name, address, contact numbers, membership, vehicle, etc).

xPIL provides a number of elements/attributes that are applicable to a person only (e.g. gender, marital status, age, ethnicity, physical information, hobbies, etc)

xPIL provides a number of elements/attributes that are applicable to an organisation only (e.g. industry type, registration details, number of employees, etc)

6.1 Reuse of xNL and xAL Structure for Person or Organisation Name and Address

“Name” of *xPIL* schema reuses *PartyNameType* constructs from *xNL* namespace and “Address” of the *xPIL* schema reuses *AddressType* construct from *xAL* namespace as illustrated in the following diagram:


FreeTextLines
Free text description of the party as line 1, line 2, line n.
n:PartyNameType
PartyName
a:AddressType
Addresses
Address
1..∞
A container for all party addresses
Accounts
A container to define the accounts details of the party such as utility account, financil accounts

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

6.2 Party Structures - Examples

The following examples illustrate use of a selection of party constructs.

6.2.1 Example – Qualification Details

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

6.2.2 Example – Birth Details

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

6.2.3 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:Type="DocumentID">74183768C</p:DocumentElement>
   <p:DocumentElement p:Type="DocumentType">Driver License</p:DocumentElement>
   <p:DocumentElement p:Type="Priviledge">Silver</p:DocumentElement>
   <p:DocumentElement p:Type="Restriction">Car</p:DocumentElement>
</p:Document>
```

6.2.4 Example – Contact Phone Number

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

6.2.5 Example – Electronic Address Identifiers

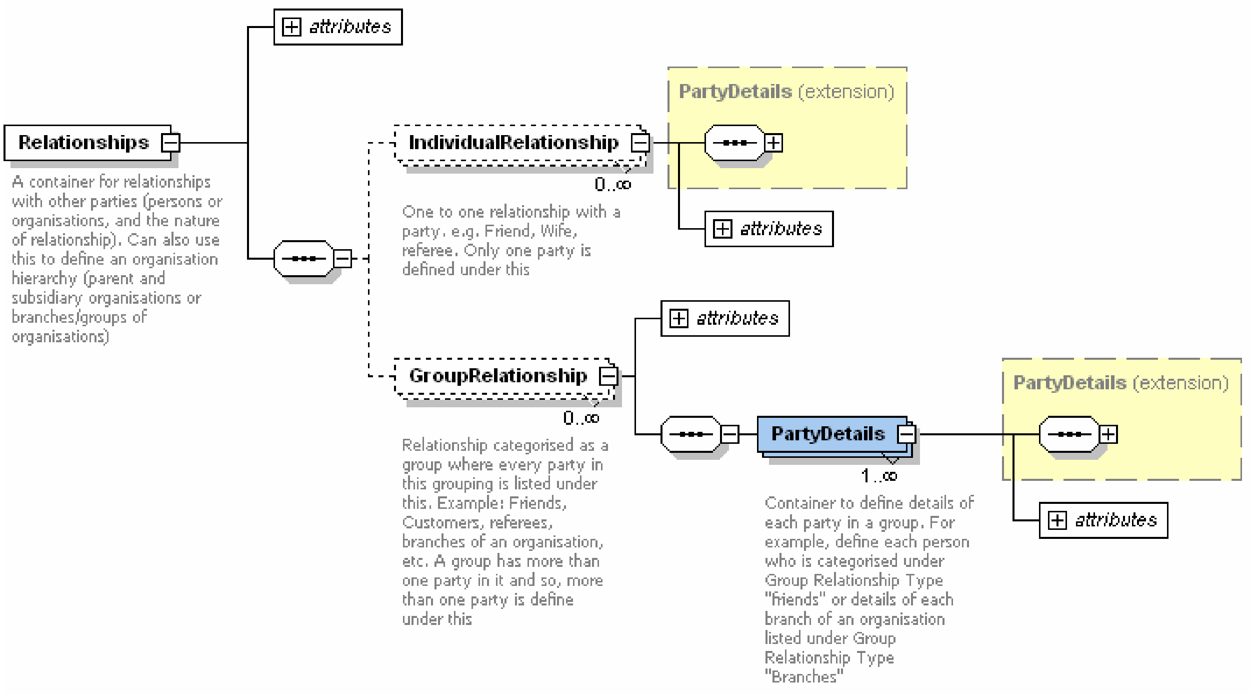
```
<p:ElectronicAddressIdentifiers>
   <p:ElectronicAddressIdentifier p:Type="SKYPE" p:Usage="Personal">rkumar
   </p:ElectronicAddressIdentifiers>
   <p:ElectronicAddressIdentifier p:Type="EMAIL" p:Usage="Business">ram.kumar@email.com
   </p:ElectronicAddressIdentifiers>
   <p:ElectronicAddressIdentifier p:Type="URL"
p:Usage="Personal">http://www.ramkumar.com
   </p:ElectronicAddressIdentifiers>
```

6.3 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).

6.4 Representing Relationships with other Parties

xPIL provides the ability to also define relationships between a party (person or an organisation) and other parties (person or organisation). This is shown in the following diagram (an extract of XML schema):


attributes
Relationships
A container for relationships with other parties (persons or organisations, and the nature of relationship). Can also use this to define an organisation hierarchy (parent and subsidiary organisations or branches/groups of organisations)
IndividualRelationship
0..∞
One to one relationship with a party. e.g. Friend, Wife, referee. Only one party is defined under this
PartyDetails (extension)
attributes
attributes
GroupRelationship
0..∞
Relationship categorised as a group where every party in this grouping is listed under this. Example: Friends, Customers, referees, branches of an organisation, etc. A group has more than one party in it and so, more than one party is define under this
PartyDetails
1..∞
Container to define details of each party in a group. For example, define each person who is categorised under Group Relationship Type "friends" or details of each branch of an organisation listed under Group Relationship Type "Branches"
PartyDetails (extension)
attributes

Two categories of relationships with a party (Person or Organisation) can be defined. They are

- Individual Relationship, and
- Group Relationship

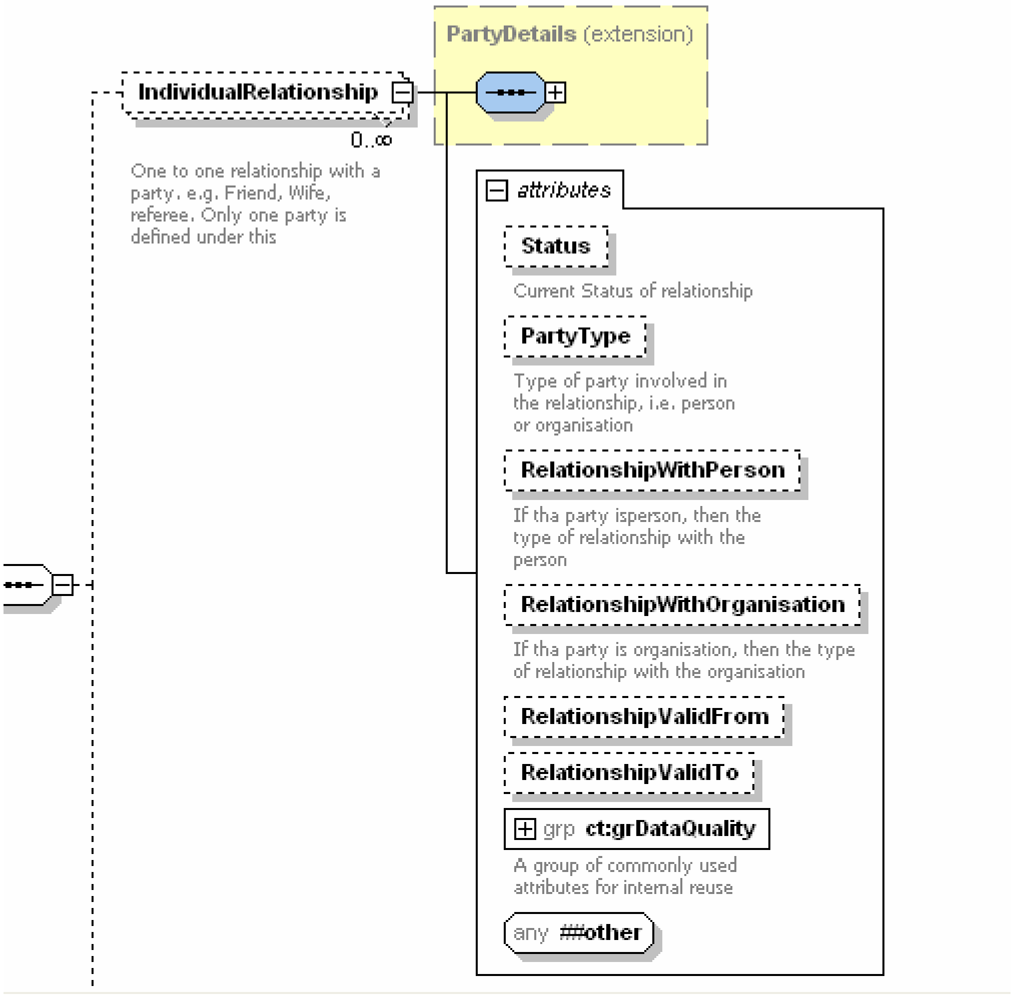
Individual Relationship is a one on one relationship with another party. Examples include, Friend, Spouse, Referee, Contact, etc for a person, and Client, customer, branch, head office, etc for an organisation.

Group Relationship is categorisation of a group of parties together. For example, friends, contacts, referees, relatives, children, etc. for a person, and clients, customers, branches, subsidiaries, partners, etc for an Organisation.

Details of each party can be defined namely, Person Name, Organisation Name, Contact Numbers and Electronic Address Identifiers.

6.4.1 Individual Relationship

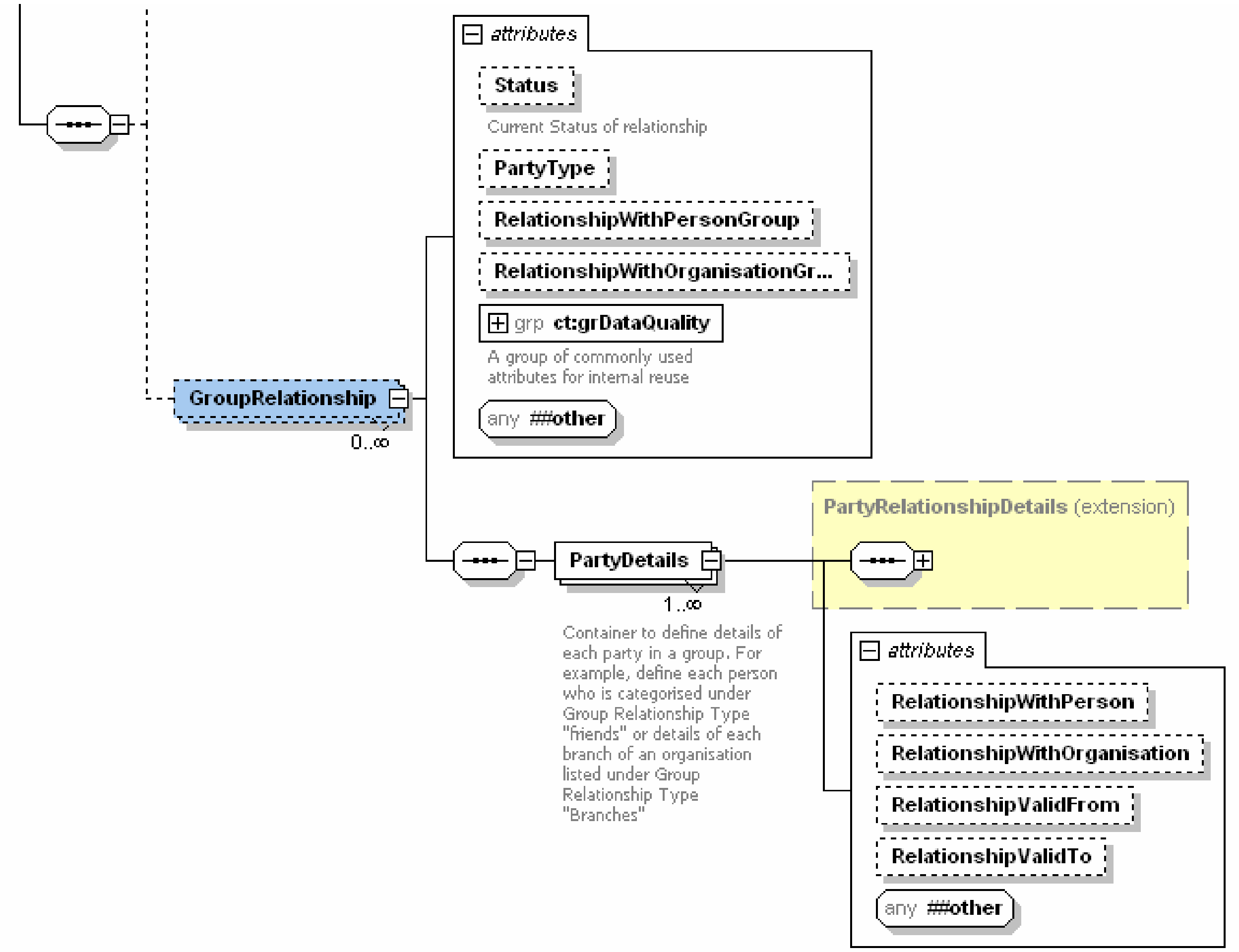
Details of individual relationship are shown in the figure below:


PartyDetails (extension)
IndividualRelationship
0..∞
One to one relationship with a party, e.g. Friend, Wife, referee. Only one party is defined under this
attributes
Status
Current Status of relationship
PartyType
Type of party involved in the relationship, i.e. person or organisation
RelationshipWithPerson
If tha party isperson, then the type of relationship with the person
RelationshipWithOrganisation
If tha party is organisation, then the type of relationship with the organisation
RelationshipValidFrom
RelationshipValidTo
grp ct:grDataQuality
A group of commonly used attributes for internal reuse
any ##other

The attribute *Status* defines the status of relationship; attribute *RelationshipWithPerson* defines the type of relationship with the person (e.g. friend, spouse) if the party is a person; attribute *RelationshipWithOrganisation* defines the type of relationship with the organisation (e.g. client, branch, subsidiary) if the party is an organisation; attributes *RelationshipValidFrom* and *RelationshipValidTo* defines the dates of the relationship with the party.

6.4.2 Group Relationship

Details of group relationship are shown in the figure below:


attributes
Status
Current Status of relationship
PartyType
RelationshipWithPersonGroup
RelationshipWithOrganisationGr...
grp ct:grDataQuality
A group of commonly used attributes for internal reuse
any ##other
GroupRelationship
0..∞
PartyRelationshipDetails (extension)
PartyDetails
1..∞
Container to define details of each party in a group. For example, define each person who is categorised under Group Relationship Type "friends" or details of each branch of an organisation listed under Group Relationship Type "Branches"
attributes
RelationshipWithPerson
RelationshipWithOrganisation
RelationshipValidFrom
RelationshipValidTo
any ##other

The attribute *Status* defines the status of relationship; if the party is of type *person*, attribute *RelationshipWithPersonGroup* defines the grouping of the persons into a type of relationship such as *Friends, Relatives, Children, Referees, Customers*; if the party is of type *organisation*, attribute *RelationshipWithOrganisationGroup* defines the grouping of organisations into a type of relationship such as *Branches, Subsidiaries, Partners, Customers*, etc.

Under the *PartyDetails* element, each party associated with the group is defined (details of the party namely, name, address and contact details). If the party is a *person* and let us say, the *RelationshipWithPersonGroup* value is *children*. Then, the attribute *RelationshipWithPerson* under *PartyDetails* element can be used to define the type of child such as *daughter*, *brother*, etc.

If the party is an *organisation* and let us say, the *RelationshipWithOrganisationGroup* value is *Partners*. Then, the attribute *RelationshipWithOrganisation* under *PartyDetails* element can be used to define the type of partner such as *solution partner, channel partner, marketing partner*, etc. The attributes *RelationshipValidFrom* and *RelationshipValidTo* defines the dates of the relationship with the party.

6.4.3 Example – Person Relationship with other Persons of type “Friends”

```
<p:Relationships>
   <p:GroupRelationship p:RelationshipWithPersonGroup=”Friends”>
      <p:PartyDetails>
          <p:PersonName>
             <p:NameElement=”FullName”>Andy Chen</NameElement>
          </p:PersonName>
      </p:PartyDetails>
      <p:PartyDetails>
          <p:PersonName>
             <p:NameElement=”FullName”>John Freedman</NameElement>
          </p:PersonName>
      </p:PartyDetails>
      <p:PartyDetails>
          <p:PersonName>
             <p:NameElement=”FullName”>Peter Jackson</NameElement>
          </p:PersonName>
      </p:PartyDetails>
   </p:GroupRelationship>
</p:Relationships>
```

6.4.4 Example – Organisation Relationship with other Organisations of type “Worldwide Branches”

```
<p:Relationships>
   <p:GroupRelationship p:RelationshipWithOrganisationGroup=”Worldwide Branches”>
     <p:PartyDetails>
        <p:NameLine>XYZ Pty. Ltd</p:NameLine>
        <p:Address>
         <p:FreeTextAddress>
           <p:AddressLine>23 Archer Street, Chastwood, NSW 2067,
               Australia
            </p:AddressLine>
         </p:FreeTextAddress>
        </p:Address>
      <p:PartyDetails>
         <p:NameLine>XYZ Pte. Ltd</p:NameLine>
         <p:Address>
          <p:FreeTextAddress>
            <p:AddressLine>15, Meena Rd, K.K.Nagar, Chennai 600078
               India
            </p:AddressLine>
          </p:FreeTextAddress>
         </p:Address>
       </p:PartyDetails>
    </p:GroupRelationship>
</p:Relationships>
```

6.4.5 Example – Person Relationship with another Person

```
<p:Relationships>
   <p:IndividualRelationship p:RelationsipWithPersonGroup=”Son”>
      <p:PersonName>
         <p:NameElement=”FullName”>Andy Chen</NameElement>
      </p:PersonName>
   </p:IndividualRelationship>
</p:Relationships>
```

6.5 Data Types

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

All free-text values of elements (text nodes) and attributes are constrained by a simple type "*String*" (255 characters in size and collapsed white spaces) defined in *CommonTypes.xsd*. Other XML Schema data types are also used throughout the schema.

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

6.6 Code Lists (Enumerations)

Use of code lists/enumerations is identical to use of code lists for entity "*Name*". Refer to section 3.3 for more information.

Code lists/enumerations used in *xPIL* for code list option 1 reside in an "include" *xPIL-types.xsd*. Code lists/enumerations used in *xPIL* for code list option 2 reside as .gc genericode files.

NOTE: The code list/enumeration values for different code lists/enumeration lists that are provided as part of the specifications are not complete. They only provides some sample values and it is up to the end users to customise them to meet their data exchange requirements if the default values are incomplete, not appropriate or over kill

6.7 Order of Elements and Presentation

Order of elements without qualifier (@...type attribute) MUST be preserved for correct presentation as described in section 3.6.

6.8 Data Mapping

Mapping data between *xPIL* schema and a database is similar to that of entity "*Name*" as described in section 3.7.

6.9 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 3.8 for more information.

6.10 Extensibility

All elements in *Party* namespaces are extensible as described in section 3.10.

6.11 Linking and Referencing

All linking and referencing rules described in section 3.9 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 3.11.

6.12 Schema Conformance

Schema conformance described in section 3.12 is fully applicable to entity “*Party*”.

6.13 Schema Customization Guidelines

Schema customisation rules and concepts described in section 3.13 are fully applicable to entity “*Party*”.

6.13.1 Customizing the Code Lists/Enumerations of Party

If there is no intent to use the code list/enumeration list for the *xPIL* schema elements, the code list/enumeration list can be ignored. There is no absolute must rule that the default values for the enumeration lists provided by the specification must exist. The list can be empty also. As long as the code list/enumeration list values are agreed between the parties involved in data exchange (whether data exchange between internal business system or with external systems), interoperability is not an issue.

In Option 1 of representing code lists, the values clarifying the meaning of party element types (e.g. *DocumentType,ElectronicAddressIdentifierType*) in *xPIL.xsd* were intentionally taken out of the main schema file into an “include” file (*xPIL-types.xsd*) to make customisation easier. In Option 2 of Code List representation, these code lists are represented as separate .gc file in genericode format.

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

NOTE: The code lists values for different code list/enumeration lists that are provided as part of the specification are not complete. They only provides some sample values and it is up to the end users to customise them to meet their data exchange requirements if the default values are incomplete, not appropriate or over kill

6.13.1.1 End User Customised Code List - An Example

In the example below, we use *Identifier* element of *xPIL.xsd.* The default values provided by CIQ Specification for *Identifier* type’s enumeration are given below. The user might want to restrict these values. So, the user can customise the code list for *Identifier* types by making the *PartyIdentifierTypeEnumeration* with the required values as shown in the table below.

Original xPIL values for PartyIdentifierTypeList	Possible end user customised values
TaxID	TaxID
CompanyID	
NationalID	
RegistrationID	

This level of flexibility allows some customization of the schema through changing the code list/enumerations only, without changing the basic structure of the schema. It is important to ensure that all schema users involved in data exchange use the same cod list/enumerations for interoperability to be successful. This has to be negotiated between the data exchange parties and a proper governance process SHOULD be in place to manage this process.

6.13.1.2 Implications of changing Party Entity Schema

Any changes to the Party Entity schema (*xPIL.xsd*) are likely to break the compatibility one way or another.

It MAY be possible that an XML fragment created for the original schema is invalid for the altered schema or vice versa. This issue needs to be considered before making any changes to the schema that could break the compatibility.

6.13.2 Using the Code list Methodology (UMCLVV) to customize Party Schema to meet application specific requirements

The other approach to customize the CIQ party schema (*xPIL.xsd*) without touching it is by using the UMCLVV. In this approach, one can use Schematron patterns to define assertion rules to customize party schema without touching or modifying it. For example, it is possible to customize party schema to restrict the use of party entities (elements and attributes) that are not required for a specific application. These entities can be restricted using Schematron based assertion rules.

NOTE: The business rules used to constraint CIQ party schema SHOULD be agreed by all the parties that are involved in data exchange of CIQ based party data to ensure interoperability and the rules SHOULD be governed.

7 Differences between two types of Entity Schemas for CIQ Specifications

CIQ Specifications comes with two types of entity schemas (*xNL.xsd, xAL.xsd, xPIL.xsd*, and *xNAL.xsd*) based on the type of code lists/enumerations used. The types of code lists/enumerations options used are:

Option1 (Default): All code lists for an entity represented using XML schema (in one file) and “included” in the appropriate entity schema (*xNL-types.xsd, xAL-types.xsd*, *xNAL-types.xsd,* and *xPIL-types.xsd*).

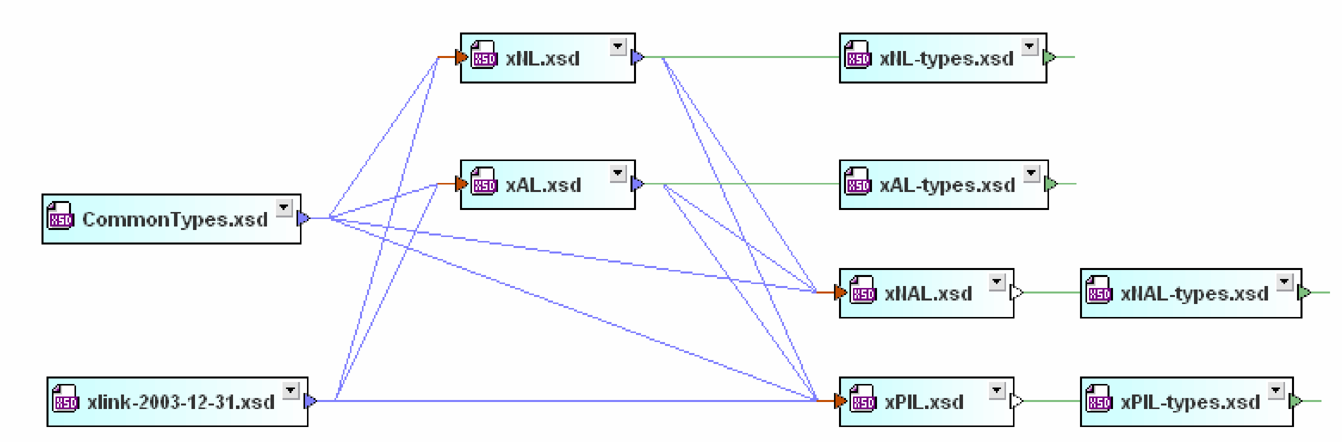
Option 2: Code Lists represented using Genericode structure of OASIS Codelist TC. Each enumeration list in option 1 is a separate “.gc” file in this option.

7.1 Files for Option 1 (The Default)

Following are the XML schema files provided as default in CIQ Specifications package for Option 1:

- *xNL.xsd*
- *xNL-types.xsd (**10** Default Code Lists defined for xNL)*
- *xAL.xsd*
- *xAL-types.xsd (**30** Default Code Lists defined for xAL)*
- *xPIL.xsd*
- *xPIL-types.xsd (**56** Default Code Lists defined for xPIL)*
- *xNAL.xsd*
- *xNAL-types.xsd (**1** Default Code List defined for xNAL)*
- *CommonTypes.xsd (**2** Default Code Lists defined for Common Type for all entities)*
- *xlink-2003-12-21.xsd*

The relationship between the different XML Schemas for Option 1 is shown in the following diagram:



7.2 Files for Option 2

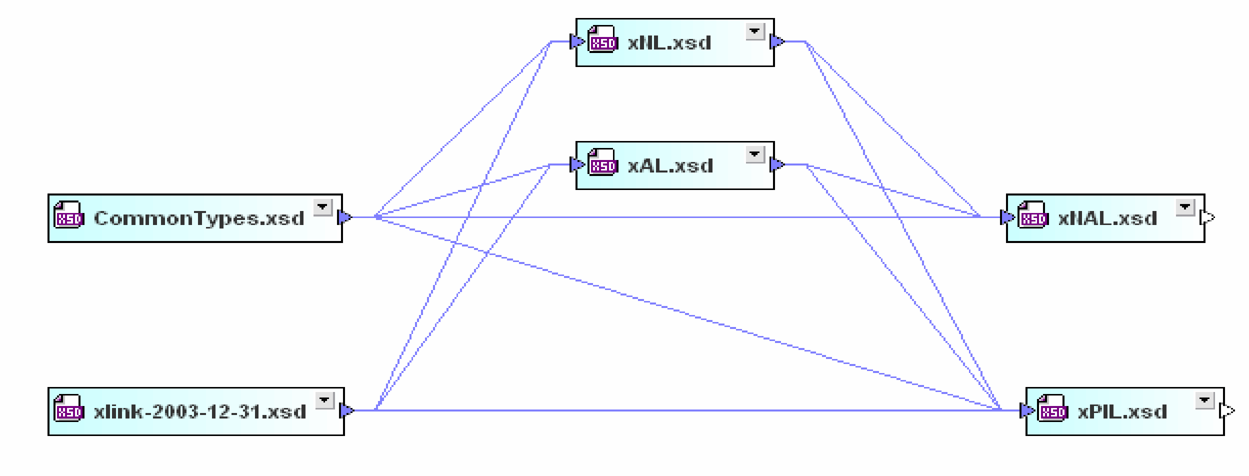
Following are the files provided as default in CIQ Specifications package for Option 2:

7.2.1 XML Schema Files

- *xNL.xsd*
- *xAL.xsd*
- *xPIL.xsd*
- *xNAL.xsd*
- *CommonTypes.xsd*
- *xlink-2003-12-21.xsd*

No **-types.xsd* files exist in Option 2 as all the code lists are defined as genericode files.

The relationship between the different schemas for Option 2 is shown in the following figure. As you can see, the enumeration list XML schemas do not exist. Instead, each CIQ entity (Name, Address, and Party) has a set of genericode based Code List files (.gc).



7.2.2 Genericode Based Code List Files

7.2.2.1 For Name (xNL)

10 default genericode based code list files with .gc extension. Each enumeration list in Option 1 is defined as a separate file in Option 2.

7.2.2.2 For Address (xAL)

30 default genericode based code list files with .gc extension. Each enumeration list in Option 1 is defined as a separate file in Option 2.

7.2.2.3 For Name and Address (xNAL)

1 default genericode based code list file with .gc extension. The enumeration list in Option 1 is defined as a separate file in Option 2.

7.2.2.4 For Party (xPIL)

56 default genericode based code list files with .gc extension. Each enumeration list in Option 1 is defined as a separate file in Option 2.

7.2.2.5 For Common Types

2 default genericode based code list files with .gc extension.

7.3 Namespace Assignment

Both the types of entity schemas (for option 1 and option 2) use the same namespaces to ensure that the XML instance documents generated from any of these two options are compatible with both types of CIQ entity XML schemas.

7.4 The Difference in Entity Schemas

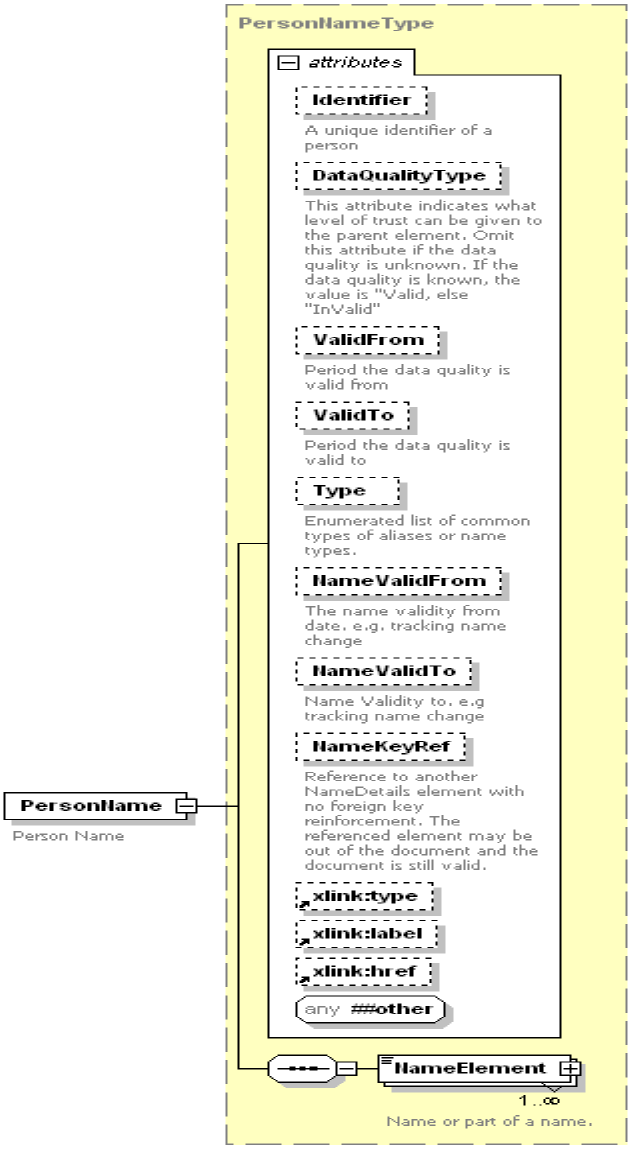
The key difference between the two types of entity schemas are the additional metadata information for information item values in XML instances for Option 2. This metadata information is defined as OPTIONAL attributes. It is not mandatory to have instance level metadata, but having it allows an instance to disambiguate a code value that might be the same value from two different lists. An application interpreting a given information item that has different values from different lists MAY need the user to specify some or all of the list metadata from which the value is found, especially if the value is ambiguous.

Four types metadata attributes are used in Option 2 entity schema attributes that reference code lists and they are:

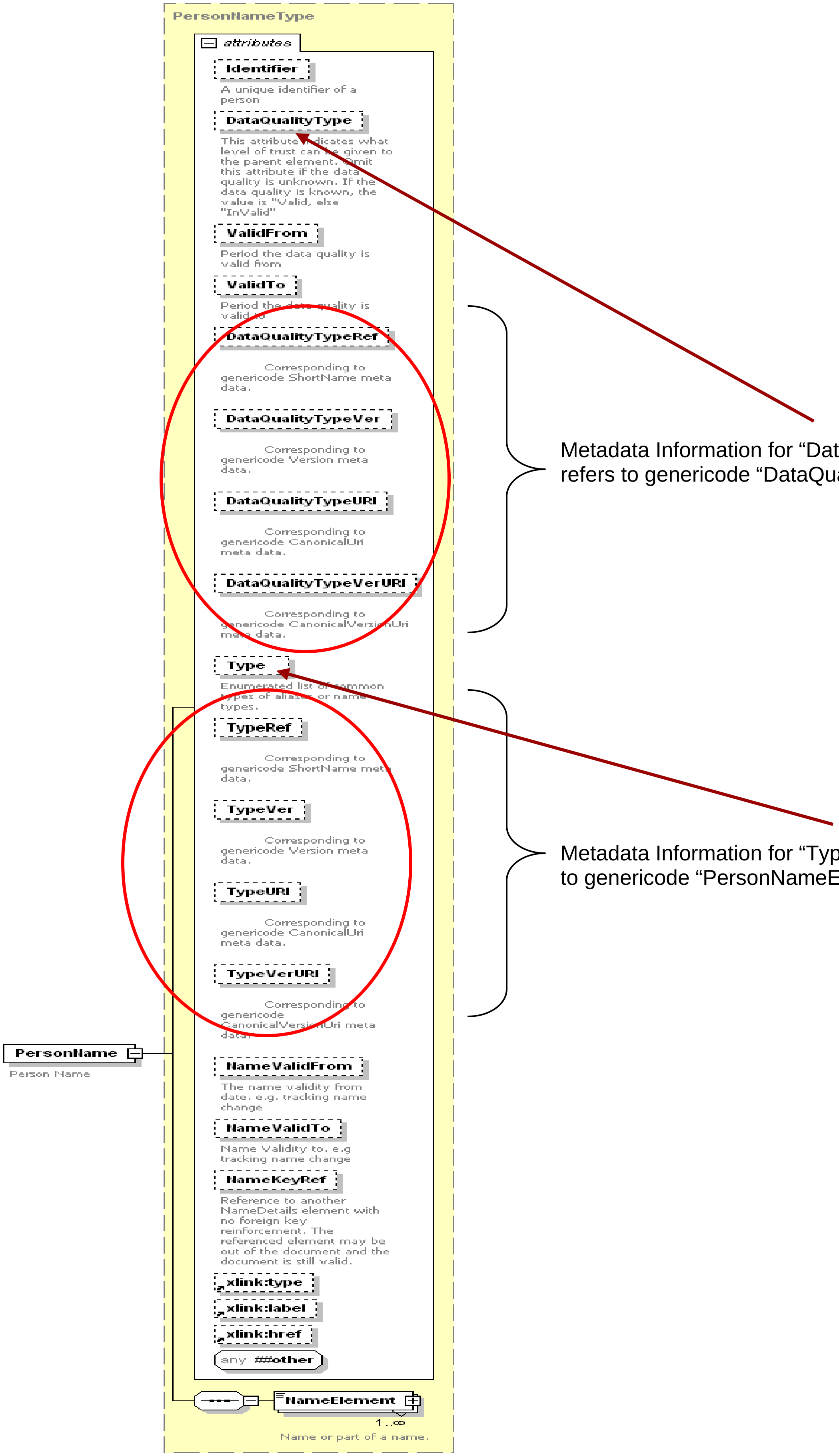
- *Ref* – corresponds to genericode <ShortName> reference
- *Ver* – corresponds to genericode <Version> version of the file
- *URI* – corresponds to genericode <CanonicalUri> abstract identifier for all versions of the code list
- *VerURI* – corresponds to genericode <CanonicalVersionUri> abstract identifier for this version of the code list

For detailed explanation of metadata information, read the Code List Value Validation methodology document (http://www.oasis-open.org/committees/document.php?document_id=21324)

The figure below shows "PersonName" element in Option 1 (using *xNL-types.xsd* for all Name entity associated code lists) of *xNL.xsd*:


PersonNameType
attributes
Identifier
A unique identifier of a person
DataQualityType
This attribute indicates what level of trust can be given to the parent element. Omit this attribute if the data quality is unknown. If the data quality is known, the value is "Valid, else "InValid"
ValidFrom
Period the data quality is valid from
ValidTo
Period the data quality is valid to
Type
Enumerated list of common types of aliases or name types.
NameValidFrom
The name validity from date. e.g. tracking name change
NameValidTo
Name Validity to. e.g tracking name change
NameKeyRef
Reference to another NameDetails element with no foreign key reinforcement. The referenced element may be out of the document and the document is still valid.
xlink:type
xlink:label
xlink:href
any ##other
PersonName
Person Name
NameElement
1..∞
Name or part of a name.

The figure below shows *PersonName* element in Option 2 (using genericode for Name entity associated code lists) of *xNL.xsd* with metadata information for genericode based code lists:


PersonNameType
attributes
Identifier
A unique identifier of a person
DataQualityType
This attribute indicates what level of trust can be given to the parent element. Omit this attribute if the data quality is unknown. If the data quality is known, the value is "Valid, else "InValid"
ValidFrom
Period the data quality is valid from
ValidTo
Period the data quality is valid to
DataQualityTypeRef
Corresponding to genericode ShortName meta data.
DataQualityTypeVer
Corresponding to genericode Version meta data.
DataQualityTypeURI
Corresponding to genericode CanonicalUri meta data.
DataQualityTypeVerURI
Corresponding to genericode CanonicalVersionUri meta data.
Type
Enumerated list of common types of aliases or name types.
TypeRef
Corresponding to genericode ShortName meta data.
TypeVer
Corresponding to genericode Version meta data.
TypeURI
Corresponding to genericode CanonicalUri meta data.
TypeVerURI
Corresponding to genericode CanonicalVersionUri meta data.
PersonName
Person Name
NameValidFrom
The name validity from date. e.g. tracking name change
NameValidTo
Name Validity to. e.g tracking name change
NameKeyRef
Reference to another NameDetails element with no foreign key reinforcement. The referenced element may be out of the document and the document is still valid.
xlink:type
xlink:label
xlink:href
any ##other
NameElement
1..∞
Name or part of a name.

Metadata Information for “DataQualityType” attribute that refers to genericode “DataQualityEnumeration.gc” file

Metadata Information for “Type” attribute that refers to genericode “PersonNameEnumeration.gc” file

7.4.1 Compatibility between XML documents produced from the two options

An XML instance document produced from Option 1 SHOULD be valid against the CIQ entity XML schemas of Option 2.

Similarly, an XML document produced from Option 2 SHOULD be valid against the CIQ entity XML schemas of Option 1 provided the metadata information in Option 2 are not used as those metadata attributes do not exist in Option 1 XML schemas.

8 Data Exchange and Interoperability

It is the view of the CIQ committee that to enable interoperability of data/information between parties, the best solution is to parse the data elements into its atomic elements thereby preserving the semantics and quality of data. By this way the parties involved in data exchange will be in the best position to understand the semantics and quality of data which minimises interoperability issues. How the data will be exchanged between parties, whether in parsed or unparsed structure, must be negotiated between the parties to enable interoperability.

One cannot expect interoperability to occur automatically without some sort of negotiation between parties (e.g. Information Exchange Agreement, whether internal or external to an organisation) involved in data exchange. Once information exchange agreements between parties are in place, then the data/information exchange process can be automated. Moreover, the entire information exchange and interoperability process SHOULD be managed through an effective governance process which SHOULD involve all the parties involved in the information exchange process. This enables effective and efficient management of any change to the information exchange process in the future.

8.1 Data Interoperability Success Formula

We at OASIS CIQ TC strongly believe in the following “Data Interoperability Success Formula”:

Data Interoperability = Open Data Architecture + Data Integration + Data Quality + Data Standards + Data Semantics + Data Governance

All components on the right hand side of the above formula are important for successful data interoperability. The term “Open” used here indicates artifacts that are independent of any proprietary solution (e.g. open industry artifacts or artifacts that are open within an enterprise).

8.2 Information Exchange Agreement - Guidelines

To ensure interoperability of CIQ represented data/information between applications/business systems (whether internal to the organization or external to the organization), it is strongly advised that an information exchange agreement/specification for CIQ SHOULD is in place. This agreement/specification SHOULD outline in detail the customization of CIQ specifications.

Following are the features of CIQ specifications that assist in customization of the specifications to meet specific application or data exchange requirements, and the details of customization SHOULD be documented and agreed (if involving more than one party in data exchange) at application/system design time to enable automating interoperability of information/data represented using CIQ specifications at application/system run time:

- List of all elements of CIQ XML Schemas that SHOULD be used in the exchange. This includes details of which elements are mandatory and which elements are OPTIONAL
- List of all attributes of CIQ XML Schemas that SHOULD be used in the exchange. This includes details of which attributes are mandatory and which attributes are OPTIONAL
- The approach that will be used for Code Lists (Option 1 or Option 2)
- The code list values that SHOULD be used for each CIQ code lists. This includes updating the default XML Schemas for code lists (Option 1) with the values to be used and updating the default genericode based code lists (Option 2) with the values to be used. These code list files SHOULD then be implemented by all applications/systems involved in data exchange. If genericode based code list approach (Option 2) is used, then the XSLTs for value validation SHOULD be generated and implemented by all applications/systems involved in data exchange.
- Whether xLink or Key Reference SHOULD be used to reference party, name or address, and the details

- Whether XML schema SHOULD be extended by using new attributes from a non-target namespace and if so, details of the additional attributes
- Whether business rules SHOULD be defined to constrain the CIQ XML schemas and if so, details of the business rules that SHOULD be implemented consistently by all applications/systems involved in data exchange

Once the agreement is implemented, it is vital that the agreement SHOULD governed through a governance process to manage change effectively and efficiently. All parties involved in the data exchange process SHOULD be key stakeholders of the governance process.

9 Miscellaneous

9.1 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.

9.2 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.

9.3 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.

10 Conformance

The keywords "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", “MAY” and "OPTIONAL" interpreted as described in [RFC2119] are used as the conformance clauses throughout this document.

10.1 Conformance Clauses

10.1.1 Specifications Schema Conformance

Implementation of CIQ Specifications namely the XML Schemas (*xNL.xsd, xAL.xsd, xNAL.xsd*, and *xPIL.xsd*) MUST conform to the specifications if the implementation conforms to as stated in section 3.12.

10.1.2 Specifications Schema Extensibility Conformance

Implementation of CIQ Specifications namely the XML Schemas (*xNL.xsd, xAL.xsd, xNAL.xsd*, and *xPIL.xsd*) by extending them MUST conform as stated in section 3.9.

10.1.3 Specifications Code List Schema Customization Conformance

Customization of the Code List XML Schemas (*xNL-types.xsd, xAL-types.xsd, xNAL-types.xsd*, and *xPIL-types.xsd*) using Option 1 MUST be well formed. Changes to the default values provided as part of the specifications is OPTIONAL and MAY be modified by the user.

10.1.4 Interoperability Conformance

Implementation of CIQ Specifications between two or more applications/systems or parties helps achieve interoperability if the implementation conforms to using the agreed conformance clauses as defined in sections 10.1.4.1, 10.1.4.2, 10.1.4.3, 10.1.4.4, 10.1.4.5, and 10.1.4.6.

10.1.4.1 Interoperability Conformance - Using Elements and Attributes

Implementation of elements and attributes of CIQ XML Schema enables interoperability if the following conditions are agreed by two or more parties involved in data exchange and are met:

1. The OPTIONAL elements in the XML Schema that SHOULD be used for implementation and the OPTIONAL elements in the XML Schema that SHOULD be ignored. See section 8.2.
2. The OPTIONAL attributes in the XML Schema that SHOULD be used for implementation and the OPTIONAL attributes in the XML Schema that SHOULD be ignored. See section 8.2 .

10.1.4.2 Interoperability Conformance - Extending the Schema

Implementation of the CIQ schema by extending it SHOULD be agreed and managed between two or more parties involved in the data exchange and MUST be conformed to in order to achieve interoperability as stated in section 3.9.

10.1.4.3 Interoperability Conformance - Using Code Lists

Implementation of a Code List approach SHOULD be agreed and conformance to the selected approach between two or more parties involved in the data exchange MUST be achieved in order to ensure interoperability and this is stated in section 3.4.

10.1.4.4 Interoperability Conformance - Customizing the Code Lists

Implementation of the Code List values SHOULD be agreed between two or more parties involved in the data exchange and MUST be conformed to as agreed in order to ensure interoperability as stated in section 3.4.

10.1.4.5 Interoperability Conformance - Customizing the Schema

Customization of the schema SHOULD be achieved by the following ways:

1. Using Code List values
2. Defining new business rules to constraint the schema

Implementation of the above approaches SHOULD be agreed between two or more parties involved in the data exchange and MUST be conformed to in order to achieve interoperability as stated in section 3.13.

10.1.4.6 Interoperability Conformance - Data/Information Exchange Agreement

Implementation and conformance of the implementation to the agreed Data/Information Exchange Agreement between two or more parties involved in the data exchange MUST be achieved to ensure interoperability as stated in section 8.2.

A. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

John Glaubitz	Vertex, Inc	Member, CIQ TC
Max Voskob	Individual	Former Member, CIQ TC
Hidajet Hasimbegovic	Individual	Member, CIQ TC
Robert James	Individual	Member, CIQ TC
Joe Lubenow	Individual	Member, CIQ TC
Mark Meadows	Microsoft Corporation	Former Member, CIQ TC
John Putman	Individual	Former Member, CIQ TC
Michael Roytman	Vertex, Inc	Member, CIQ TC
Colin Wallis	New Zealand Government	Member, CIQ TC
David Webber	Individual	Member, CIQ TC
Graham Lobsey	Individual	Member, CIQ TC
George Farkas	XBI Software, Inc	Member, CIQ TC

OASIS CIQ Technical Committee (TC) also wishes to acknowledge contributions from former members of the TC since its inception in 2000. Also, the TC would like to express its sincere thanks to the public in general (this includes other standard groups, organizations and end users) for their feedback and comments that helped the TC to improve the CIQ specifications.

Special thanks to Mr.Hugh Wallis, Director of Standards Development of extensible Business Reporting Language (xBRL) International Standards Group (http://www.xbrl.org) for working closely with the CIQ TC in jointly implementing W3C xLink specification that is now used by both xBRL and CIQ Specifications to enable interoperability between the two specifications.

Special thanks to Mr.Carl Reed, Chief Technology Officer of Open Geospatial Consortium (OGC – http://www.opengeospatial.org) for his guidance and assistance to the TC in referencing the work of OGC on GeoRSS and Geo-Coordinates for addresses/locations as part of CIQ Address Specifications.

Special thanks to Mr.Ken Holman, Chair of OASIS Code List TC (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=codelist) for his assistance to the TC in releasing the OASIS Code List version of CIQ V3.0 XML Schemas.

Last but not least, the TC thanks all users of the CIQ TC specifications in real world and for their continuous feedback and support.

B. 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.

[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.

xLink-2003-12-31.xsd

This schema was provided by the xBRL group in December 2006.

C. Revision History

Revision	Date	Editor	Changes Made
V3.0 PRD 01	13 April 2006	Ram Kumar and Max Voskob	Prepared 60 days public review draft from Committee Draft 01
V3.0 PRD 02	15 June 2007	Ram Kumar	Prepared second round of 60 days public review draft from Committee Draft 02 by including all public review comments from PRD 01. Also included is implementation of OASIS Code list specification