



Customer Information Quality Specifications Version 3.0

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

Committee Specification 02

20 September 2008

Specification URIs:

This Version:

<http://docs.oasis-open.org/ciq/v3.0/cs02/specs/ciq-specs-v3-cs2.html>
<http://docs.oasis-open.org/ciq/v3.0/cs02/specs/ciq-specs-v3-cs2.doc>
<http://docs.oasis-open.org/ciq/v3.0/cs02/specs/ciq-specs-v3-cs2.pdf>

Previous Version:

<http://docs.oasis-open.org/ciq/v3.0/prd03/specs/ciq-specs-v3.html>
<http://docs.oasis-open.org/ciq/v3.0/prd03/specs/ciq-specs-v3.doc>
<http://docs.oasis-open.org/ciq/v3.0/prd03/specs/ciq-specs-v3.pdf>

Latest Version:

<http://docs.oasis-open.org/ciq/v3.0/cs02/specs/ciq-specs-v3-cs2.html>
<http://docs.oasis-open.org/ciq/v3.0/cs02/specs/ciq-specs-v3-cs2.doc>
<http://docs.oasis-open.org/ciq/v3.0/cs02/specs/ciq-specs-v3-cs2.pdf>

Technical Committee:

OASIS Customer Information Quality

Chair:

Ram Kumar (kumar.sydney@gmail.com)

Editor:

Ram Kumar (kumar.sydney@gmail.com)

Related work:

This version of the CIQ specification replaces or supercedes OASIS CIQ V3.0 Committee Specification released in November 2007

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. This specification replaces the earlier version of the committee specifications released in November 2007.

This specification also includes errata to OASIS CIQ V3.0 xAL schema (both for default code list and genericcode approaches). The errata to xAL V3.0 schema is documented as "OASIS CIQ v3.0 xAL Schema (xAL.xsd) Errata.doc" under "supp" directory of the specification package. This is the only change in this specification compared to the V3.0 committee specifications released in November 2007.

Status:

This document was last revised or approved by the OASIS CIQ Technical Committee (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 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 www.oasis-open.org/committees/ciq.

Notices

Copyright © OASIS® 1993–2008. All Rights Reserved. OASIS trademark, IPR and other policies apply.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

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", "Genericcode", and "UBL" are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organisation 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, PARTY AND PARTY RELATIONSHIP	8
1.1	TERMINOLOGY	8
1.2	DEFINITIONS.....	8
2	CIQ SPECIFICATIONS VERSION 3.0.....	9
2.1	CIQ VERSION 3.0 COMMITTEE SPECIFICATION RELEASED IN NOVEMBER 2007	9
2.2	FORMAL DESIGN REQUIREMENTS.....	9
2.3	MAJOR CIQ SPECIFICATION ENTITIES	9
2.4	VERSION 3.0 XML SCHEMA FILES	10
2.5	COMMON DESIGN CONCEPTS USED.....	11
2.6	NAMESPACES USED	11
2.7	OTHER OPEN INDUSTRY SPECIFICATIONS/STANDARDS USED.....	12
3	ENTITY “NAME” (EXTENSIBLE NAME LANGUAGE).....	13
3.1	SEMANTICS OF “NAME”.....	13
3.1.1	<i>Example 1 – No Semantics (Unstructured/Free Text Data)</i>	<i>14</i>
3.1.2	<i>Example 2 – Minimal Semantics (Partially Structured Data)</i>	<i>14</i>
3.1.3	<i>Example 3 – Full Semantics (Fully Structured Data).....</i>	<i>15</i>
3.2	DATA TYPES.....	15
3.3	CODE LISTS (ENUMERATIONS)	15
3.3.1	<i>What is a Code List?.....</i>	<i>15</i>
3.3.2	<i>The importance of Code Lists for CIQ Specifications</i>	<i>16</i>
3.3.2.1	<i>Example.....</i>	<i>16</i>
3.3.3	<i>Customisable Code Lists.....</i>	<i>17</i>
3.3.3.1	<i>Example.....</i>	<i>17</i>
3.3.4	<i>Improving Interoperability using Code Lists.....</i>	<i>18</i>
3.4	USING CODE LISTS IN CIQ SPECIFICATIONS – TWO OPTIONS	18
3.4.1	<i>Why Two Options.....</i>	<i>18</i>
3.4.2	<i>Option 1 – “Include” Code Lists (The Default Approach).....</i>	<i>19</i>
3.4.2.1	<i>Code List Representation (Option 1) – An Example</i>	<i>19</i>
3.4.2.2	<i>Customising Code Lists (Option 1) – An Example.....</i>	<i>20</i>
3.4.2.3	<i>Code List Use (Option 1) Example – Point-to-Point</i>	<i>20</i>
3.4.2.4	<i>Code List Use (Option 1) Example – Locale Specific</i>	<i>21</i>
3.4.3	<i>Option 2 – Code Lists using Genericcode Approach.....</i>	<i>21</i>
3.4.3.1	<i>Code List (Option 2) Value Validation using Context Value Association</i>	<i>21</i>
3.4.3.2	<i>Two Pass Value Validation (Option 2)</i>	<i>22</i>
3.4.3.3	<i>Code List Representation in Genericcode (Option 2) – An Example</i>	<i>23</i>
3.4.3.4	<i>Customising Genericcode based Code Lists (Option 2)</i>	<i>24</i>
3.4.3.5	<i>CIQ Specifications used as a case study by OASIS Code List TC</i>	<i>24</i>
3.4.3.6	<i>References for Option 2.....</i>	<i>24</i>
3.5	CODE LIST PACKAGES – OPTION 1 AND OPTION 2	25
3.6	ORDER OF ELEMENTS AND PRESENTATION.....	25
3.6.1	<i>Example – Normal Order</i>	<i>25</i>
3.7	DATA MAPPING	25
3.7.1	<i>Example – Complex-to-simple Mapping.....</i>	<i>26</i>
3.7.2	<i>Example – Simple-to-complex Mapping</i>	<i>26</i>
3.8	DATA QUALITY	27
3.8.1	<i>Example – Data Quality</i>	<i>27</i>
3.8.2	<i>Data Quality Verification and Trust.....</i>	<i>27</i>
3.8.3	<i>Data Validation</i>	<i>28</i>
3.9	EXTENSIBILITY	28
3.9.1	<i>Extending the Schemas to Meet Application Specific Requirements</i>	<i>28</i>
3.9.2	<i>Extensibility - Practical Applications.....</i>	<i>28</i>

3.9.2.1 System-specific Identifiers.....	28
3.9.2.2 Additional Metadata	29
3.10 LINKING AND REFERENCING.....	29
3.10.1 Using xLink [OPTIONAL].....	29
3.10.2 Using Key Reference [OPTIONAL].....	30
3.11 ID ATTRIBUTE	30
3.12 SCHEMA CONFORMANCE.....	31
3.13 SCHEMA CUSTOMISATION GUIDELINES	31
3.13.1 Namespace.....	31
3.13.2 Reducing the Entity Schema Structure.....	31
3.13.2.1 Implications of changing Name Entity Schema.....	32
3.13.3 Customising the Code Lists/Enumerations of Name.....	32
3.13.4 Using the Methodology to customise Name Schema to meet application specific requirements.....	32
3.13.4.1 Constraining Name Schema using CVA – An Example	33
4 ENTITY “ADDRESS” (EXTENSIBLE ADDRESS LANGUAGE).....	34
4.1 SEMANTICS OF “ADDRESS”.....	34
4.1.1 Example – Minimal Semantics (Unstructured/Free Text Data).....	34
4.1.2 Example – Partial Semantics (Partially Structured Data)	34
4.1.3 Example – Full Semantics (Fully Structured Data).....	36
4.2 DATA TYPES.....	36
4.3 CODE LISTS (ENUMERATIONS)	36
4.4 ORDER OF ELEMENTS AND PRESENTATION.....	36
4.5 DATA MAPPING	37
4.5.1 Example – Normal Order	37
4.6 DATA QUALITY	37
4.7 EXTENSIBILITY	37
4.8 LINKING AND REFERENCING.....	37
4.9 ID ATTRIBUTE	37
4.10 SCHEMA CONFORMANCE.....	37
4.11 ADDRESS/LOCATION REFERENCED BY GEORSS AND COORDINATES.....	38
4.11.1 Using GeoRSS in xAL Schema.....	38
4.11.1.1 GeoRSS - Example	40
4.11.1.2 GeoRSS GML – Example.....	41
4.11.2 Defining Location Coordinates in xAL Schema.....	41
4.12 SCHEMA CUSTOMISATION GUIDELINES	42
4.12.1 Customising the Code Lists/Enumerations of Address	42
4.12.1.1 End User Customised Code List - An Example.....	42
4.12.1.2 Implications of changing Address Entity Schema	43
4.12.2 Using CVA to customise CIQ Address Schema to meet application specific requirements	43
4.12.2.1 Constraining CIQ Address Schema using CVA – Example 1.....	43
4.12.2.2 Constraining CIQ Address Schema using CVA – Example 2.....	44
5 COMBINATION OF “NAME” AND “ADDRESS” (EXTENSIBLE NAME AND ADDRESS LANGUAGE).....	45
5.1 USE OF ELEMENT XNAL:RECORD	45
5.1.1 Example	45
5.2 USE OF ELEMENT XNAL:POSTALLABEL	46
5.2.1 Example	46
5.3 CREATING YOUR OWN NAME AND ADDRESS APPLICATION SCHEMA.....	47
6 ENTITY “PARTY” (EXTENSIBLE PARTY INFORMATION LANGUAGE).....	48
6.1 REUSE OF xNL AND xAL STRUCTURE FOR PERSON OR ORGANISATION NAME AND ADDRESS	48
6.2 PARTY STRUCTURES - EXAMPLES.....	49
6.2.1 Example – Qualification Details.....	49
6.2.2 Example – Birth Details.....	49
6.2.3 Example – Driver License.....	49
6.2.4 Example – Contact Phone Number.....	49

6.2.5 Example – Electronic Address Identifiers.....	49
6.3 DEALING WITH JOINT PARTY NAMES	50
6.4 REPRESENTING RELATIONSHIPS WITH OTHER PARTIES	50
6.4.1 Example – Person Relationship with other Persons of type “Friend”	52
6.4.2 Example – Organisation Relationship with other Organisations of type “Branch”	52
6.4.3 Example – Person Relationship with another Person.....	52
6.5 DATA TYPES.....	53
6.6 CODE LISTS (ENUMERATIONS)	53
6.7 ORDER OF ELEMENTS AND PRESENTATION.....	53
6.8 DATA MAPPING	53
6.9 DATA QUALITY	53
6.10 EXTENSIBILITY	53
6.11 LINKING AND REFERENCING.....	53
6.12 SCHEMA CONFORMANCE.....	54
6.13 SCHEMA CUSTOMISATION GUIDELINES	54
6.13.1 Customising the Code Lists/Enumerations of Party	54
6.13.1.1 End User Customised Code List - An Example.....	54
6.13.1.2 Implications of changing Party Entity Schema	54
6.13.2 Using CVA to customise Party Schema to meet application specific requirements	55
7 DIFFERENCES BETWEEN TWO TYPES OF ENTITY SCHEMAS FOR CIQ SPECIFICATIONS....	56
7.1 FILES FOR OPTION 1 (THE DEFAULT).....	56
7.2 FILES FOR OPTION 2.....	57
7.2.1 XML Schema Files.....	57
7.2.2 Genericcode Based Code List Files.....	57
7.2.2.1 For Name (xNL)	57
7.2.2.2 For Address (xAL).....	57
7.2.2.3 For Name and Address (xNAL).....	57
7.2.2.4 For Party (xPIL).....	57
7.2.2.5 For Common Types	58
7.3 NAMESPACE ASSIGNMENT.....	58
7.4 DIFFERENCES BETWEEN CIQ ENTITY SCHEMAS USED IN OPTION 1 AND OPTION 2.....	58
7.4.1 Compatibility between XML documents produced using Option 1 and Option 2 CIQ XML Schemas	61
7.4.2 Which Code List Package to Use? Option 1 or Option 2?	61
8 DATA EXCHANGE AND INTEROPERABILITY	62
8.1 DATA INTEROPERABILITY SUCCESS FORMULA	62
8.2 INFORMATION EXCHANGE AGREEMENT – GUIDELINES	62
9 CONFORMANCE.....	64
9.1 CONFORMANCE CLAUSES	64
9.1.1 Specifications Schema Conformance.....	64
9.1.2 Specifications Schema Extensibility Conformance.....	64
9.1.3 Specifications Code List Schema Customisation Conformance.....	64
9.1.4 Interoperability Conformance	64
9.1.4.1 Interoperability Conformance – Using Elements and Attributes	64
9.1.4.2 Interoperability Conformance – Extending the Schema	64
9.1.4.3 Interoperability Conformance – Using Code Lists.....	64
9.1.4.4 Interoperability Conformance – Customising the Code Lists	64
9.1.4.5 Interoperability Conformance – Customising the Schema.....	65
9.1.4.6 Interoperability Conformance – Data/Information Exchange Agreement.....	65
10 MISCELLANEOUS	66
10.1 DOCUMENTATION	66
10.2 EXAMPLES	66
10.3 CONTRIBUTIONS FROM PUBLIC	66
11 CHANGE LOG.....	67

A.	ACKNOWLEDGEMENTS	68
B.	INTELLECTUAL PROPERTY RIGHTS, PATENTS, LICENSES AND ROYALTIES	69
C.	REVISION HISTORY	70

1 Name, Address, Party and Party Relationship

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 organisation

Address

A physical location or a mail delivery point

Party

A Party could be of two types namely,

- Person
- Organisation

An Organisation 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 organisation's name and address are generally the key identifiers (but not necessarily the unique identifiers) of a "Party". A "Customer" is of type "Party".

Party Relationship

Pair wise affiliation or association between two people, between two organisations, or between an organisation and a person.

xPRL supports chains of interlocking pair wise party relationships, linked by common members.

2 CIQ Specifications Version 3.0

2.1 CIQ Version 3.0 Committee Specification Released in November 2007

This section is only applicable to users who have downloaded and are using CIQ Version 3.0 Committee specification package (**OASIS CIQ V3.0.zip**) that was released in November 2007. This section is not applicable to new users of the CIQ Specifications.

In January 2008, a minor issue with Version 3.0 xAL schema was identified and an errata was issued. This errata is now included part of this specification. Details about the errata are documented in “**ciq-v3-change-log**” file (.doc or .pdf or .html) under “supp” directory. For existing users of CIQ V3.0 specification, this errata SHOULD have an impact and the details about the impact and how to implement the errata are documented in “**ciq-package-overview-v3**” file (.doc or .pdf or .html) under the “supp” directory.

2.2 Formal Design Requirements

Following are the formal design requirements taken into consideration for version 3.0 XML Schemas of CIQ Specifications:

- Data structures SHOULD be described using W3C XML Schema language
- Data structures SHOULD be separated into multiple namespaces for reuse of the core Party entities (e.g. Person Name, Organisation Name, Address, Party Centric Information)
- 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 application-specific information to be attached to entities without breaking the structures.
- Implementation complexity SHOULD be proportional to the complexity of the subset of data structures used by the implementer
- Data structures SHOULD be customisable to meet different end user requirements without breaking the structures and at the same time, conforming to the core specification.
- Data structures SHOULD minimise the use of mandatory element fields to provide flexibility to deal with various application and data exchange requirements and at the same time provides users the opportunity to define data constraint rules (business rules) outside of the data structures.

2.3 Major CIQ Specification Entities

The entire party information space is divided into a number of complex information types that are viewed as core entities. This enables re-use of the core entities as required. We categorise these core entities of CIQ Specifications into four namely,

- Name
- Address
- Party Centric Information, and
- Party Relationships

70 Following are the basic and core CIQ specification entities defined in XML schemas as re-usable
71 types:

- 72 • Name (Person or Organisation - see *xNL.xsd schema*)
- 73 • Address (see *xAL.xsd schema*)
- 74 • Name and Address combined (see *xNAL.xsd schema*)
- 75 • Personal details of a person (person-centric information) (see *xPIL.xsd schema*)
- 76 • Organisation specific details (organisation-centric information) (see *xPIL.xsd schema*)
- 77 • Party Relationships (see *xPRL.xsd* [not available in this release, but released separately as part of
78 *xPRL specification*] and *xLink-2003-12-31-revised.xsd schemas*)

79 These core entities are supported by relevant code lists/enumerations to add “semantics/meaning” to
80 the data they represent. This will be discussed in detail in the following sections.
81

82 2.4 Version 3.0 XML Schema Files

83 Following are the different XML schemas produced for version 3.0:

XML 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 associate/link 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 centric information
xPIL-types.xsd	Entity Party (organisation or individual) Enumerations	Defines a set of enumerations to support party centric 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	Implements a subset of W3C xLink specification attributes as XML schema

XML Schema File name	Description	Comments
*.gc files	Entity Party, Name, and Address	Defines a set of enumerations/code lists in genericcode format

84 2.5 Common Design Concepts Used

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

87 Name, Address and Party information components of version 3.0 share common design concepts that are
88 implemented as XML Schemas. This commonality should simplify understanding and adoption of the
89 XML Schemas. The xNAL schema design concept varies slightly as it is only a simple container for
90 associating/linking names and addresses.

91 The design concepts of Name, Address and Party schemas are similar in terms of the way semantic
92 information is represented to add the required “meaning” to the data. For example, for a person’s name
93 data, “Given Name, “Middle Name’ Surname” etc, are the semantic information that add meaning to the
94 data.

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

101 2.6 Namespaces Used

102 Following are the namespaces used in the specification:

Entity	Namespace	Suggested Prefix	XML 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 xPRL-types.xsd
xLink	http://www.w3.org/1999/xlink	xLink	xLink-2003-12-31.xsd

103

104

105 **2.7 Other Open Industry Specifications/Standards Used**

106 This document contains references to XML Linking Language (XLink) Version 1.0, W3C
107 Recommendation 27 June 2001 available at <http://www.w3.org/TR/xlink/> . The CIQ TC strongly
108 recommends readers to read the xLink specification from W3C if they want to use this supported feature
109 in CIQ Specifications.

110 This document contains references to Code List version 1.0, OASIS Code List Representation TC
111 Committee Specification 01, December 2007 available at <http://www.oasis-open.org/committees/codelist>.
112 The CIQ TC strongly recommends readers to read the code list specification if they want to use this
113 supported feature in CIQ Specification.

114 This document contains references to Context Value Association, Working Draft 0.4, 20 April 2008 ,
115 available at <http://www.oasis-open.org/committees/codelist>. The CIQ TC strongly recommends readers to
116 read the methodology if they want to use this supported feature in CIQ Specification.

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

120

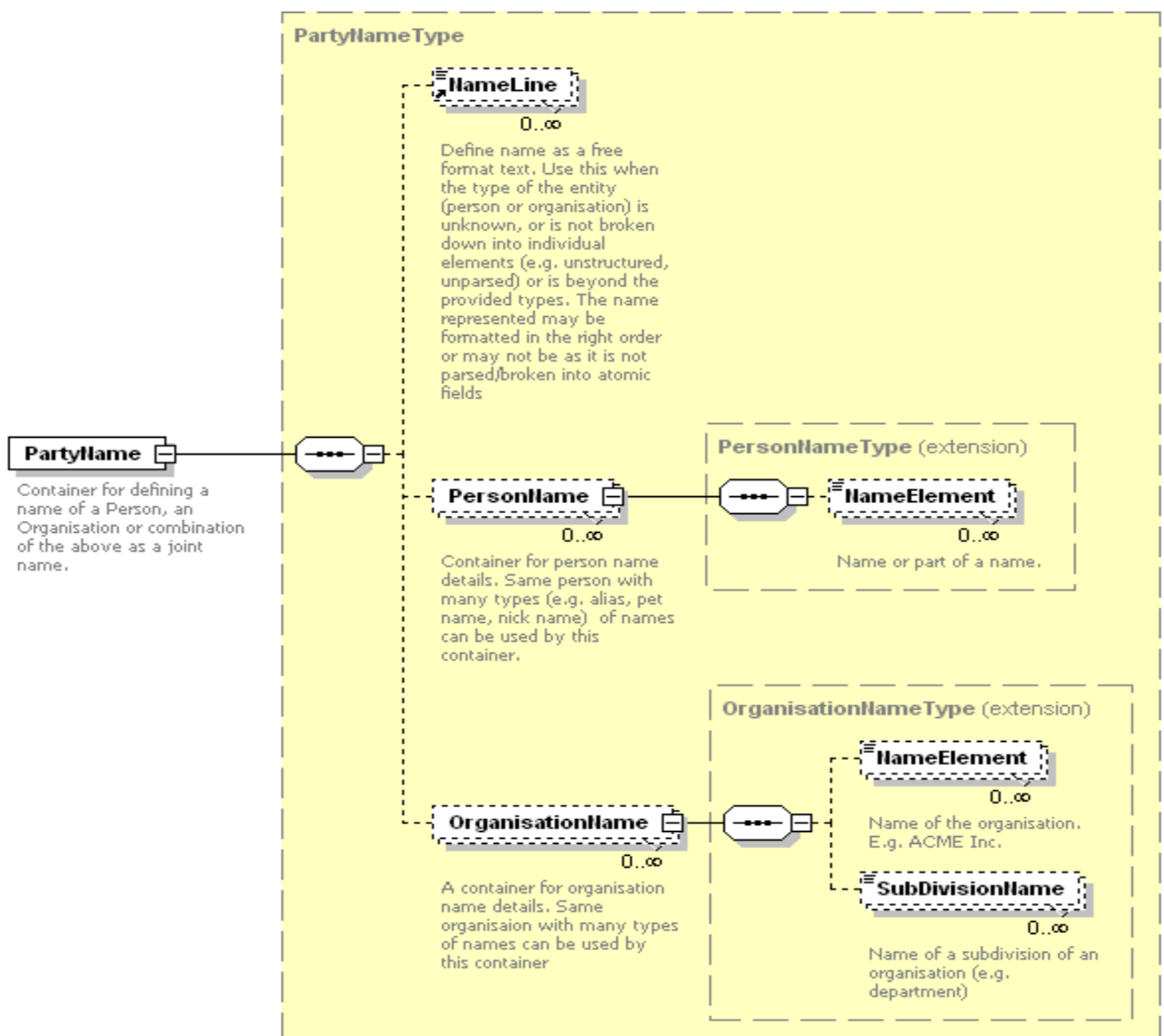
121 3 Entity “Name” (extensible Name Language)

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

124 3.1 Semantics of “Name”

125 CIQ Version 3.0 “Name” XML schema is separated into two parts: a structural part (*xNL.xsd*) as shown in
 126 the XML schema diagram below and, separate enumeration/code list files (code lists defined in an XML
 127 schema (*xNL-types.xsd*) and also, code lists represented in “Genericcode” format as .gc files) supporting
 128 the structure by adding semantics to the data. “Genericcode” will be discussed in later sections.

129 The structural part (*xNL.xsd*) SHOULD remain unchanged over the course of time while the code
 130 list/enumeration files (*xNL-types.xsd* or .gc files) MAY be customised to meet particular implementation
 131 needs as the semantics of data varies from one requirement to another.



132
 133 In the schema structure above (*xNL.xsd*), “NameElement” stores the name of a party and the supporting
 134 enumeration lists referenced as *attributes* in the schema structure (see the *xNL.xsd* schema for the list of
 135 attributes or the HTML documentation of the schema) that provide the semantic meaning of the data.

136 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)

The least level of complexity in representing party name data is when a typical database does not differentiate between a person name and an organisation name where only one field has been allocated for storing the complete 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, i.e. person name or an organisation name, and what the individual name elements (atomic data) 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. Data in this free formatted/unstructured text form is classified as “poor quality” as it is subject to different interpretations and MAY cause interoperability problems when exchanged between two or more applications/systems.

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

3.1.2 Example 2 – Minimal Semantics (Partially Structured Data)

The medium level of complexity in representing data 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. The quality of data in this case is marginally better than Example 1.

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

176
177
178
179
180
181

182 3.1.3 Example 3 – Full Semantics (Fully Structured Data)

183 The maximum level of complexity in representing data is when a database differentiates between person
184 and organisation name and also differentiates between different name elements within a name (the
185 semantics). The data is structured and the quality of data is excellent.

```
186 <n:PartyName>  
187   <n:PersonName>  
188     <n:NameElement Abbreviation="true" ElementType="Title">Mr</n:NameElement>  
189     <n:NameElement ElementType="FirstName">Jeremy</n:NameElement>  
190     <n:NameElement ElementType="MiddleName">Apatuta</n:NameElement>  
191     <n:NameElement ElementType="LastName">Johnson</n:NameElement>  
192     <n:NameElement ElementType="GenerationIdentifier">III</n:NameElement>  
193     <n:NameElement ElementType="GenerationIdentifier">Junior</n:NameElement>  
194     <n:NameElement ElementType="Title">PhD</n:NameElement>  
195   </n:PersonName>  
196 </n:PartyName>
```

197 This example introduces *ElementType* attribute that indicates the exact meaning of the name element.
198 Few applications and in particular, applications dealing with data quality and integrity, fall under this “Full
199 Semantics” category and often, the database supported by these applications are high in the quality of
200 the data it manages. This is an additional level of semantics that is supported through code
201 list/enumerated values. Technically, the enumerations sit in a separate schema (*xNL-types.xsd*) or in
202 “Genericcode” files.

203 The more structured the data is, the better the interoperability of the data.

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

```
206 <xs:simpleType name="PersonNameElementsEnumeration">  
207   <xs:restriction base="xs:string">  
208     <xs:enumeration value="PrecedingTitle"/>  
209     <xs:enumeration value="Title"/>  
210     <xs:enumeration value="FirstName"/>  
211     <xs:enumeration value="MiddleName"/>  
212     <xs:enumeration value="LastName"/>  
213     <xs:enumeration value="OtherName"/>  
214     <xs:enumeration value="Alias"/>  
215     <xs:enumeration value="GenerationIdentifier"/>  
216   </xs:restriction>  
217 </xs:simpleType>
```

218 3.2 Data Types

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

220 All free-text values of elements (text nodes) and attributes are constrained by a simple type
221 “*NormalizedString*” (collapsed white spaces) defined in *CommonTypes.xsd*. Other XML Schema data
222 types are also used throughout the schema.

223 3.3 Code Lists (Enumerations)

224 This is an important section that users MUST give serious attention if they want to customise the CIQ
225 schemas to meet their specific requirements.

226 3.3.1 What is a Code List?

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

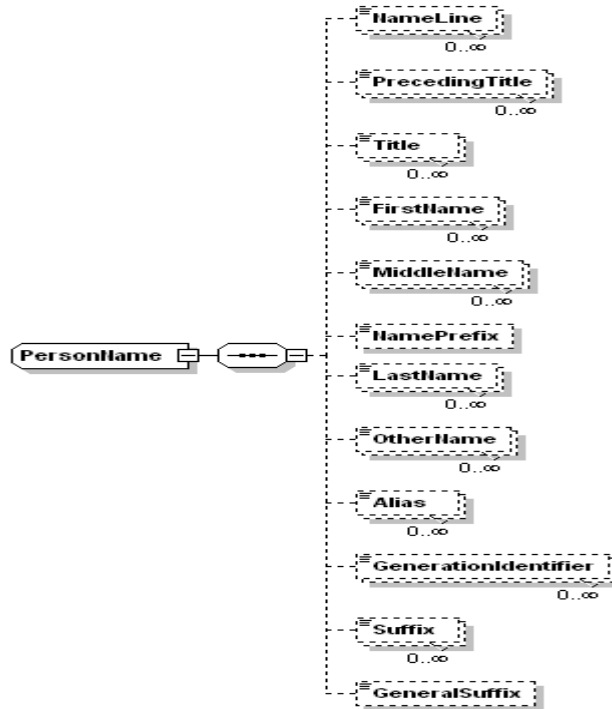
231

232 XML Schema describes the structural and lexical constraints on an XML document. Some information
233 items in a document are described in the schema lexically as a simple value whereby the value is a code

234 representing an agreed-upon semantic concept. The value used is typically chosen from a set of unique
235 coded values enumerating related concepts. These sets of coded values are sometimes termed *code*
236 *lists*.

237 3.3.2 The importance of Code Lists for CIQ Specifications

238 Earlier versions of CIQ Name, Address and Party Information specifications had concrete schema
239 grammar (e.g. First Name, Middle Name, Last Name, etc XML elements/tags for a person name as
240 shown in the figure below) to define the party entities.



241 This did not satisfy many name, address and party data usage scenarios that are geographic and cultural
242 specific. For example, in certain cultures, the concept of first name, middle name, and last/family/surname
243 for a person name does not exist. Representing person names from these cultures in the earlier version
244 of CIQ Specifications were difficult as its name schema (v2.0 of *xNL.xsd* as shown in the above figure)
245 had pre-defined element names as *FirstName*, *MiddleName*, and *LastName*, and they were semantically
246 incorrect metadata for the data. To be precise, in some culture where the concept of *First Name* does not
247 exist, using *First Name* element of CIQ specification to a data that appears in the first position of a
248 person's name string is semantically incorrect.
249

250 3.3.2.1 Example

251 Let us look at the following example (this is not a fictitious person name, but real legal name of a
252 person born in the USA, who is a childhood friend of the Chair of CIQ TC. The street name and
253 father's name of the person have been deliberately changed in this example to protect the identity
254 of the person). The person's name is:

255 *Mr. William Street Rajan United States Virginia Indian*, where

256 "*William Street*", is the name of the street where the person was born

257 "*Rajan*", is the name of the person's father

258 "*United States*", is the country where the person was born

259 "*Virginia*", is the state where the person was born, and

260 "*Indian*", is the origin of the person

261 The person is legally and formally called as "WRUVI"
262 In the above example, using the concept of First Name, Middle Name, Last Name, Surname, Family
263 Name, etc. does not provide the intended meaning of the name, and therefore, the meaning of the data is
264 lost.

265 3.3.3 Customisable Code Lists

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

274 The default code list values/enumerations for Party Name used in the CIQ Specifications are built using
275 common sense and with culture-specific view of the subject area (in this case Anglo-American culture,
276 where the terms such as *First Name*, *Middle Name*, *Last Name* are used), rather than adopted from a
277 specific application. The reason why we say "cultural specific view" is because some cultures do not have
278 the concept of *First Name*, *Middle Name*, and *Last Name* and so on.

279 **NOTE:** The code list/enumeration values for different code/enumeration lists that are
280 provided as part of the specifications are not complete. They only provide sample
281 values (and in most case no values) and it is up to the end users to customise them to
282 meet their data exchange requirements if the default values are incomplete, not
283 appropriate or over kill

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

294 Therefore, for a generic international specification like CIQ that is independent of any
295 application/industry/culture, the ability to customise the specification to define context specific semantics
296 to the data is important.

297 3.3.3.1 Example

298 Now let us revisit example 3.3.2.1 again. To overcome the semantics problem and to not loose the
299 semantics of the data, using version 3.0 of the CIQ specification, users can define the correct context
300 specific semantics to the person name data as follows:

```
301 <n:PartyName>  
302   <n:PersonName>  
303     <n:NameElement ElementType="Title">Mr.</n:NameElement>  
304     <n:NameElement ElementType="Birth Street Name">William Street</n:NameElement>  
305     <n:NameElement ElementType="Father Name">Rajan</n:NameElement>  
306     <n:NameElement ElementType="Country Of Birth">United States</n:NameElement>  
307     <n:NameElement ElementType="State Of Birth">Virginia</n:NameElement>  
308     <n:NameElement ElementType="Country Of Origin">Indian</n:NameElement>  
309   </n:PersonName>  
310 </n:PartyName>
```

311 All user has to do is include the above semantic values that do not exist in the default
312 “*PersonNameElementList*” code list (e.g. Birth Street, Father Name, Country Of Birth, State of Birth,
313 Country Of Origin) without modifying the core *xNL.xsd* schema.

314 3.3.4 Improving Interoperability using Code Lists

315 Using customisable code list approach provided by CIQ Specifications, interoperability of data
316 (represented using CIQ Specifications) between applications can be significantly improved. Any
317 attribute/element that can add semantic meaning to data (e.g. type of address, where the value “Airport”
318 adds semantic meaning to an address data) is defined as a customisable code list in CIQ Specifications.
319 For example, *PersonName* element in *xNL.xsd* uses an attribute *PersonIDType* that provides a default
320 code list, but with no default values. When a code list has no values, XML Parsers treat the
321 attribute/element that references the code list as the same XML schema data type defined for that
322 element/attribute. This allows an application to define any value for the data type without any restriction.
323 This could result in interoperability breakdown between the sending application and the receiver
324 application because the receiving application needs to know the value of the data type that is passed for
325 further processing and it is unknown at run time. To improve interoperability by controlling the use of the
326 values for the data type, users SHOULD define specific values in the code list during design time, and
327 importantly these values SHOULD be agreed at design time by the parties exchanging the data. This will
328 give confidence to the users that the data exchanged during application run time conforms to the code list
329 values that have been agreed during application design time.

330 To provide enough flexibility to users to define the semantics of the data, over 100 default code lists (most
331 of them are empty, i.e., no default code values are provided) are provided by CIQ Specifications that are
332 customisable by users to improve interoperability of data.

333 3.4 Using Code Lists in CIQ Specifications – Two Options

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

- 335 • **OPTION 1:** An XML schema file per CIQ entity (*Name*, *Address* and *Party*) representing all code lists
336 for the entity is provided as part of the specification. The enumeration/code list files are *xNL-types.xsd*
337 (for *Name* Entity code lists), *xAL-types.xsd* (For *Address* Entity code lists), *xNAL-types.xsd* (for *Name*
338 and *Address* Entities code list) and *xPIL-types.xsd* (for *Party* Entity code lists). This is the “DEFAULT”
339 approach for using code lists.
- 340 • **OPTION 2:** A “Genericcode” based code list file (.gc) per code list for all CIQ entities (*Name*, *Address*
341 and *Party*) is provided as part of the specification. “Genericcode” is an OASIS industry specification for
342 representing code lists. For example, *xNL-types.xsd* file has 13 code lists in Option 1, and these code
343 lists are represented as 13 individual genericcode (.gc) files in this option. Therefore, *xNL-types.xsd*,
344 *xAL-types.xsd*, *xNAL-types.xsd*, and *xPIL-types.xsd* Code List schemas are not part of this option and
345 instead, are replaced with .gc files.

346 Users MUST choose one of the above two options as part of the specification implementation, but MUST
347 NOT use both the options in the same implementation.

348 3.4.1 Why Two Options

349 Option 2 (Genericcode approach) uses a “two pass validation” methodology (explained in the later
350 sections) on a CIQ XML document instance (first pass for XML document structural and lexical validation
351 against the core CIQ XML schema (*xNL.xsd*) and second pass for validation of code list value in the XML
352 document).

353 CIQ specifications are normally embedded/implemented as part of any broader application specific
354 schema such as customer information management, postal services, identity management, human
355 resource management, financial services, etc. The application specific schema MAY or MAY NOT
356 implement genericcode approach to code lists. If only Option 2 is provided as part of the CIQ
357 specifications, end users implementing CIQ XML schema that is included as part of their application
358 specific schema to represent party data, will be forced to perform two pass validation on the application’s
359 XML document instance and in particular, on the fragments in the XML document where party data is

360 represented using CIQ. This limits the usage of CIQ specifications for wider adoption and hence, two
361 options are provided to enable end users to pick an approach that suits their requirements. The two
362 options are explained in detail in the following sections.

363 3.4.2 Option 1 – “Include” Code Lists (The Default Approach)

364 “Include” code lists are XML schemas that are “included” in the CIQ entity structure XML schemas, i.e.,
365 *xNL.xsd* (Name Entity schema) “includes” *xNL-types.xsd* code list schema (as shown in the sample code
366 below), *xAL.xsd* (Address Entity schema) “includes” *xAL-types.xsd* code list schema, *xNAL.xsd* schema
367 “includes” *xNAL-types.xsd* code list schema, and *xPIL.xsd* (party entity schema) “includes” *xPIL-types.xsd*
368 schema.

```
369 <?xml version="1.0" encoding="UTF-8"?>  
370 <xs:schema xmlns="urn:oasis:names:tc:ciq:xnl:3"  
371 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xlink="http://www.w3.org/1999/xlink"  
372 xmlns:ct="urn:oasis:names:tc:ciq:ct:3" targetNamespace="urn:oasis:names:tc:ciq:xnl:3"  
373 elementFormDefault="qualified" attributeFormDefault="qualified">  
374  
375 <xs:include schemaLocation="xNL-types.xsd"/> <!--code list schema included -->
```

376 Users MAY modify the code list XML schema to add or delete values depending upon their data
377 exchange requirements without modifying the structure of the CIQ entity schemas. Validation of the code
378 list values will be performed by XML parsers as part of the XML document instance validation in “one”
379 pass (i.e., XML document structure validation and the code list value validation will be performed in one
380 pass).

381 Any changes to the code list schema results in changes to the software code (e.g. java object generated
382 from *xNL.xsd* using XML Beans must be re-created) based on the entity schema as the entity schema
383 “includes” the code list schema.

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

389 3.4.2.1 Code List Representation (Option 1) – An Example

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

```
392 <xs:simpleType name=SubDivisionTypeList">  
393   <xs:annotation>  
394     <xs:documentation> A list of common types for sub divisions  
395     </xs:documentation>  
396   </xs:annotations>  
397   <xs:restriction base="xs:string">  
398     <xs:enumeration value="Department"/>  
399     <xs:enumeration value="Branch"/>  
400     <xs:enumeration value="Business Unit"/>  
401     <xs:enumeration value="School"/>  
402     <xs:enumeration value="Section"/>  
403   </xs:restriction>  
404 </xs:simpleType>
```

405
406
407
408
409
410
411

412 3.4.2.2 Customising Code Lists (Option 1) – An Example

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

Default values for “ <i>OrganisationNameTypeList</i> ” Code List	Customised values
--	-------------------

LegalName	ReportedName
NameChange	OriginalName
CommonUse	LegalName
PublishingName	
OfficialName	
UnofficialName	
Undefined	

415 The code for the specification with the original code list for “*OrganisationNameTypeList*” would look like
416 the following:

```
417 <xs:simpleType name="OrganisationNameTypeList">  
418   <xs:restriction base="xs:string">  
419     <xs:enumeration value="LegalName" />  
420     <xs:enumeration value="NameChange" />  
421     <xs:enumeration value="CommonUse" />  
422     <xs:enumeration value="PublishingName" />  
423     <xs:enumeration value="OfficialName" />  
424     <xs:enumeration value="UnofficialName" />  
425     <xs:enumeration value="Undefined" />  
426   </xs:restriction>  
427 </xs:simpleType>
```

428 The code for the new customised code list for “*OrganisationNameTypeList*” would look like the following:

```
429 <xs:simpleType name="OrganisationNameTypeList">  
430   <xs:restriction base="xs:string">  
431     <xs:enumeration value="ReportedName" />  
432     <xs:enumeration value="OriginalName" />  
433     <xs:enumeration value="LegalName" />  
434   </xs:restriction>  
435 </xs:simpleType>
```

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

441 3.4.2.3 Code List Use (Option 1) Example – Point-to-Point

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

```
446 <xs:simpleType name="PersonNameElementsList">  
447   <xs:restriction base="xs:string">  
448     <xs:enumeration value="Title" />  
449     <xs:enumeration value="FirstName" />  
450     <xs:enumeration value="MiddleName" />  
451     <xs:enumeration value="LastName" />  
452   </xs:restriction>  
453 </xs:simpleType>
```

454 3.4.2.4 Code List Use (Option 1) Example – Locale Specific

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

```
456 <xs:simpleType name="PersonNameElementList">
457   <xs:restriction base="xs:string">
458     <xs:enumeration value="Title"/>
459     <xs:enumeration value="Name"/>
460     <xs:enumeration value="FathersName"/>
461     <xs:enumeration value="FamilyName"/>
462   </xs:restriction>
463 </xs:simpleType>
```

464 Again, it is up to the implementers involved in data exchange to modify *PersonNameElementList* code list
465 in *xNL-types.xsd* file.

466 3.4.3 Option 2 – Code Lists using Genericode Approach

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

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

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

488 This Option uses a “two” pass validation methodology, whereby, the “first” pass validation, allows the XML
489 document instance to be validated for its structure and well-formedness (ensures that information items
490 are in the correct places and are correctly formed) against the entity XML schema, and the “second” pass
491 validation allows the code list values in XML document instance to be validated against the genericode
492 based code lists and this does not involve the entity schemas.

493 Any change to the genericode based code list does not require changes to the software code (e.g. java
494 object must be re-created) based on the entity schema as the entity schema reference the genericode
495 based code list.

496 3.4.3.1 Code List (Option 2) Value Validation using Context Value Association

497 OASIS Code List Technical Committee describes an approach called “Context Value Association (CVA)”
498 for using the “two” validation approach as discussed in the previous section. CVA describes the file
499 format used in a “context/value association” file (termed in short as “a CVA file”). This file format is an
500 XML vocabulary using a subset of W3C XPath 1.0 to specify hierarchical document contexts and the
501 associated controlled vocabulary of values allowed at each context. A document context specifies one or
502 more locations found in an XML document or other similarly structured hierarchy. This file format
503 specification assumes the controlled vocabulary of values is expressed in an external resource described
504 by the genericode OASIS standard.

505 Context/value association is useful in many aspects of working with an XML document using controlled
506 vocabularies. Two examples are (1) for the direction of user data entry in the creation of an XML
507 document, ensuring that only valid values are proffered in a user interface selection such as a drop-down
508 menu; and (2) for the validation of the correct use of valid values found in an XML document.

509
510 CVA enables validating code list values and supporting document types with which trading partners can
511 agree unambiguously on the sets of code lists, identifiers and other enumerated values against which
512 exchanged documents must validate. The objective of applying CVA to a set of document instances
513 being validated is to express the lists of values that are allowed in the context of information items found
514 in the instances. One asserts that particular values must be used in particular contexts, and the validation
515 process confirms the assertions do not fail.

516 3.4.3.2 Two Pass Value Validation (Option 2)

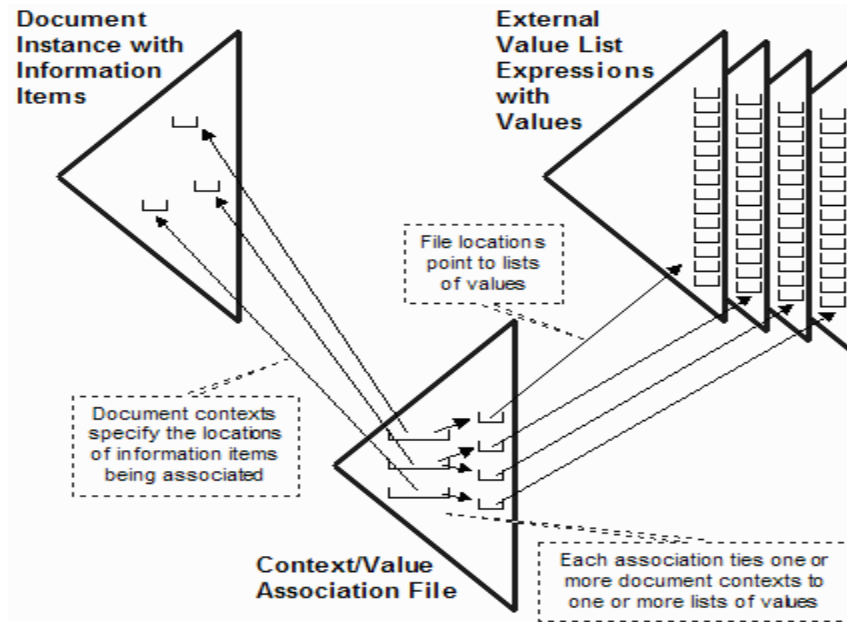
517 Schemata describe the structural and lexical constraints on a document. Some information items in a
518 document are described in the schema lexically as a simple value whereby the value is a code
519 representing an agreed-upon semantic concept. The value used is typically chosen from a set of unique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

523
524 The “first pass” can be accomplished with an XML document schema language such as W3C Schema or
525 ISO/IEC 19757-2 RELAX NG; “the second pass” is accomplished with a transformation language such as
526 a W3C XSLT 1.0 stylesheet or a Python program. In this specification, the second pass is an
527 implementation of ISO/IEC 19757-3 Schematron schemas that are utilised by CVA.

528
529 ISO Schematron is a powerful and yet simple assertion-based schema language used to confirm the
530 success or failure of a set of assertions made about XML document instances. One can use ISO
531 Schematron to express assertions supporting business rules and other limitations of XML information
532 items so as to aggregate sets of requirements for the value validation of documents.

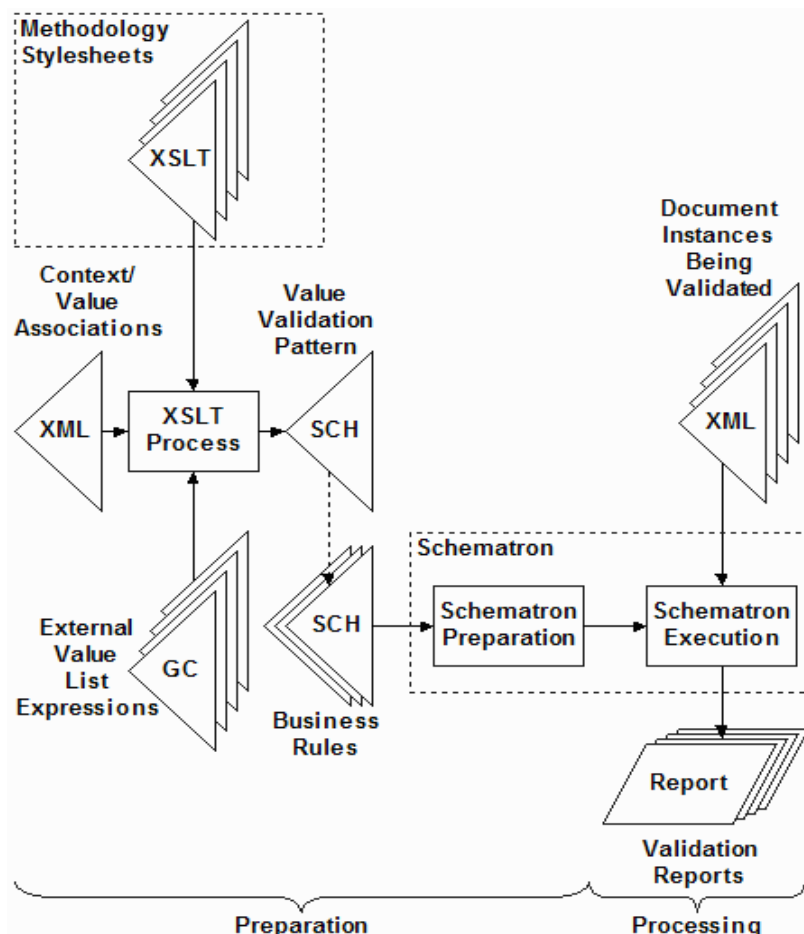
533
534 In the figure below, “Methodology context association” depicts a file of context/value associations in the
535 lower centre, where each association specifies for information items in the document instance being
536 validated which lists of valid values in external value list expressions are to be used.

537



538

539 The synthesis of a pattern of ISO Schematron assertions to validate the values found in document
 540 contexts, and the use of ISO Schematron to validate those assertions are illustrated in “Methodology
 541 overview” figure below.



542
 543 To feed the ISO Schematron process, one needs to express the contexts of information items and the
 544 values used in those contexts. CVA prescribes an XML vocabulary to create instances that express such
 545 associations of values for contexts. The stylesheets provided with CVA read these instances of
 546 context/value associations that point to externally-expressed lists of values and produce an ISO
 547 Schematron *pattern* of assertions that can then be combined with other patterns for business rule
 548 assertions to aggregate all document value validation requirements into a single process. The validation
 549 process is then used against documents to be validated, producing for each document a report of that
 550 document's failures of assertions.

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

554 3.4.3.3 Code List Representation in Genericcode (Option 2) – An Example

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

```

557 <CodeList>
558   <SimpleCodeList>
559     <Row>
560       <Value ColumnRef="code">
561         <SimpleValue>Department</SimpleValue> <!-- code list value ->
562       </Value>
563       <Value ColumnRef="name">
564         <SimpleValue>Department</SimpleValue> <!-- description of the value->
  
```

```

565     </Value>
566 </Row>
567 <Row>
568     <Value ColumnRef="code">
569         <SimpleValue>Division</SimpleValue>
570     </Value>
571     <Value ColumnRef="name">
572         <SimpleValue>Division</SimpleValue>
573     </Value>
574 </Row>
575
576 <<Row>
577     <Value ColumnRef="code">
578         <SimpleValue>Branch</SimpleValue>
579     </Value>
580     <Value ColumnRef="name">
581         <SimpleValue>Branch</SimpleValue>
582     </Value>
583 </Row>
584 <Row>
585     <Value ColumnRef="code">
586         <SimpleValue>BusinessUnit</SimpleValue>
587     </Value>
588     <Value ColumnRef="name">
589         <SimpleValue>BusinessUnit</SimpleValue>
590     </Value>
591 </Row>
592 <Row>
593     <Value ColumnRef="code">
594         <SimpleValue>Section</SimpleValue>
595     </Value>
596     <Value ColumnRef="name">
597         <SimpleValue>Section</SimpleValue>
598     </Value>
599 </Row>
600 </SimpleCodeList>
601 </CodeList>

```

602

603 3.4.3.4 Customising Genericcode based Code Lists (Option 2)

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

609 The process of customising the code lists is documented in CVA for code list and value validation.

610 3.4.3.5 CIQ Specifications used as a case study by OASIS Code List TC

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

615 3.4.3.6 References for Option 2

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

- 618 • OASIS Codelist Representation (Genericcode) Version 1.0, Committee Specification 01, December
619 2007, <http://www.oasis-open.org/committees/codelist>
- 620 • Context Value Association, Working Draft 0.4, April 2008, [http://www.oasis-](http://www.oasis-open.org/committees/codelist)
621 [open.org/committees/codelist](http://www.oasis-open.org/committees/codelist)

- 622 • OASIS Code List Adaptation Case Study (OASIS CIQ), 2007, [http://www.oasis-](http://www.oasis-open.org/committees/codelist)
623 [open.org/committees/codelist](http://www.oasis-open.org/committees/codelist)

624 3.5 Code List Packages – Option 1 and Option 2

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

631 The CIQ Specification entity schemas (*xNL.xsd*, *xAL.xsd*, *xPIL.xsd*, and *xNAL.xsd*) for both option 1 and
632 2 are in the same namespaces as users will use one of the two.

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

635 Section 7.4 explains the differences between the CIQ Core Entity schemas used in Option 1 and Option
636 2.

637 3.6 Order of Elements and Presentation

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

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

642 3.6.1 Example – Normal Order

643 `Mr Jeremy Apatuta Johnson PhD`

644 could be presented as follows

```
645 <n:PartyName>  
646   <n:PersonName>  
647     <n:NameElement>Mr</n:NameElement>  
648     <n:NameElement>Jeremy</n:NameElement>  
649     <n:NameElement>Apatuta</n:NameElement>  
650     <n:NameElement>Johnson</n:NameElement>  
651     <n:NameElement>PhD</n:NameElement>  
652   </n:PersonName>  
653 </n:PartyName>
```

654 and restored back to Mr Jeremy Apatuta Johnson PhD.

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

657 3.7 Data Mapping

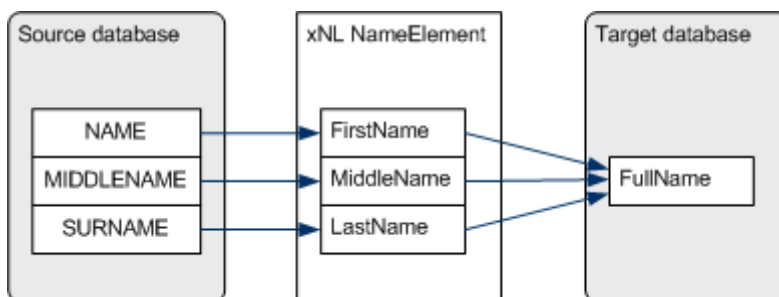
658 Mapping data between the *xNL* schema and a target database is not expected to be problematic as *xNL*
659 provides enough flexibility for virtually any level of data decomposition. However, the main issue lies in
660 the area of mapping a data provider with a data consumer through *xNL*.

661 For example, consider a data provider that has a person name in one line (free text and unparsed) and a
662 data consumer that has a highly decomposed data structure for a person's name requires first name,
663 family name and title to reside in their respective fields. There is no way of strong the free text and
664 unparsed data in the target data structure without parsing it first using some smart name parsing data
665 quality parsing/scrubbing tool (there are plenty in the market). Such parsing/scrubbing is expected to be
666 the responsibility of the data consumer under this scenario and importantly, agreeing in advance with the
667 data provider that the incoming data is not parsed.

668 **3.7.1 Example – Complex-to-simple Mapping**

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

671



672

673

674 **Source Database**

NAME	MIDDLENAME	SURNAME
John	Anthony	Jackson

675

676 **xNL**

677

```

678 <n:PersonName>
679   <n:NameElement n:ElementType="FirstName">John</n:NameElement>
680   <n:NameElement n:ElementType="MiddleName">Anthony</n:NameElement>
681   <n:NameElement n:ElementType="LastName">Jackson</n:NameElement>
682 </n:PersonName>
  
```

683

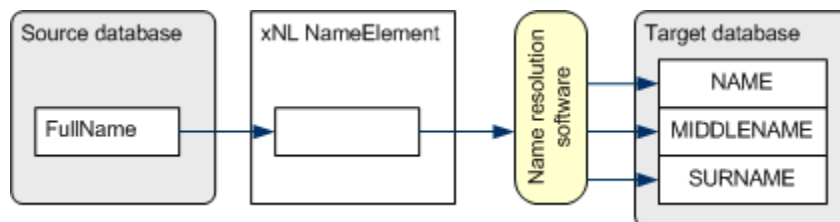
684 **Target Database**

FULLNAME
John Anthony Jackson

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

687 **3.7.2 Example – Simple-to-complex Mapping**

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



690

691

692

693

694

695

696 **Source Database**

FULLNAME

John Anthony Jackson

697

698 **xNL**

699
700
701

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

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

704

705 **Target Database**

NAME	MIDDLENAME	SURNAME
John	Anthony	Jackson

706 **3.8 Data Quality**

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

713 xNL schema allows for data quality information to be provided as part of the entity using an attribute
714 *DataQuality* that can be set to either “*Valid*” or “*Invalid*” (default values), if such status is known. If
715 *DataQuality* attribute is omitted, it is presumed that the validity of the data is unknown. Users can
716 customise the *DataQuality* code list to add more data quality attributes (e.g. confidence levels) if required.

717 *DataQuality* attribute refers to the content of a container, e.g. *PersonName*, asserting that all the values
718 are known to be true and correct in a particular defined period. This specification also has provision to
719 define partial data quality where some parts of the content are correct and some are not or unknown.

720 **3.8.1 Example – Data Quality**

721
722
723
724

```
<n:PersonName n:DataQuality="Valid"
  n: ValidFrom="2001-01-01T00:00:00"
  <n:NameElement>John Anthony Jackson</n:NameElement>
</n:PersonName>
```

725 In this example *John Anthony Jackson* is known to be the true and correct value asserted by the sender
726 of this data and the validity of the data has been recorded as of 2001-01-01.

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

729 **3.8.2 Data Quality Verification and Trust**

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

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

733 3.8.3 Data Validation

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

736 3.9 Extensibility

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

740 All elements share the same declaration:

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

742 Although this specification provides an extensibility mechanism, it is up to the participants of the data
743 exchange process to agree on the use of any extensions to the target namespace. Extensions without
744 agreements between parties involved in data exchange will break interoperability.

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

747 3.9.1 Extending the Schemas to Meet Application Specific Requirements

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

755 3.9.2 Extensibility - Practical Applications

756 3.9.2.1 System-specific Identifiers

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

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

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

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

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

770

771

772

773

774 3.9.2.2 Additional Metadata

775 Sometimes it MAY be required to include some additional metadata that is specific to a particular system
776 or application. Consider these examples:

```
777 <n:PartyName xmlns:x="urn:acme.org:corporate" x:OperatorID="buba7">  
778 .....
```

```
779  
780 <n:PartyName xmlns:b="urn:acme.org:corporate ">  
781 <n:PersonName>  
782 <n:NameElement b:Corrected="true">John Johnson</n:NameElement>  
783 </n:PersonName>  
784 </n:PartyName>
```

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

787 An example is provided as part of this specification package under xPIL examples to demonstrate the use
788 of sections 3.9.2.1 and 3.9.2.2.

789 3.10 Linking and Referencing

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

- 795 - Using *xLink*
- 796 - Using Key Reference

797 3.10.1 Using xLink [OPTIONAL]

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

803 Names can be referenced internally (i.e. within some XML infoset that contains both referencing and
804 referenced elements) through *xlink:href* pointing at an element with *xml:id* with a matching value. External
805 entities can also be referenced if they are accessible by the recipient via HTTP(s)/GET.

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

```
808 <a:Contacts  
809   xmlns:a="urn:acme.org:corporate:contacts"  
810   xmlns:n="urn:oasis:names:tc:ciq:xsd:schema:xNL:3.0/20050427"  
811   xmlns:xlink="http://www.w3.org/1999/xlink">  
812   <n:PartyName xlink:href="http://example.org/party?id=123445" xlink:type="locator"/>  
813   <n:PartyName xlink:href="http://example.org/party?id=83453485" xlink:type="locator"/>  
814 </a:Contacts>
```

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

818 Usage of *xLink* attributes in the CIQ specifications MAY slightly differ from the original *xLink* specification.
819 See *CIQ TC Party Relationships Specification* for more information on using *xLink* with xNL [Not available
820 in this version, but available as part of xPRL specification]. The *xLink* specification is available at
821 <http://www.w3.org/TR/xlink/>.

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

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

826 3.10.2 Using Key Reference [OPTIONAL]

827 This approach MAY be used for internal references (i.e. within some XML infoset that contains both
828 referencing and referenced elements). Two keys are used to reference an entity namely, *Party* and
829 *Address*. Two keys are:

- 830 1. *Key* – Primary Key of the entity, and
- 831 2. *KeyRef* – Foreign Key to reference an entity

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

```
834 <c:Customers  
835   xmlns:c="urn:acme.org:corporate:customers"  
836   xmlns:a="urn:oasis:names:tc:ciq:xl:3"  
837   xmlns:n="urn:oasis:names:tc:ciq:xnl:3"  
838   xmlns:p="urn:oasis:names:tc:ciq:xpil:3"  
839   <p:Party PartyKey="111">  
840     <n:PartyName>  
841       <n:PersonName>  
842         <n:NameElement n:ElementType="FirstName">Ram</n:NameElement>  
843         <n:NameElement n:ElementType="LastName">Kumar</n:LastName>  
844       </n:PersonName>  
845     </n:PartyName>  
846     <p:Party p:PartyKey="222">  
847       <n:PartyName>  
848         <n:PersonName>  
849           <n:NameElement n:ElementType="FirstName">Joe</n:NameElement>  
850           <n:NameElement n:ElementType="LastName">Sullivan</n:LastName>  
851         </n:PersonName>  
852       </n:PartyName>  
853     </p:Party>  
854     <c:Contacts>  
855       <c:Contact c:PartyKeyRef="222">  
856       <c:Contact c:PartyKeyRef="111">  
857     </c:Contacts>  
858 </c:Customers>
```

859 3.11 ID Attribute

860 Attribute *ID* is used with complex type *PersonNameType* and elements *PersonName* and
861 *OrganisationName*. This attribute allows unique identification of the collection of data it belongs to. The
862 value of the attribute MUST be unique within the scope of the application of xNL and the value MUST be
863 globally unique. The term ‘globally unique’ means a unique identifier that is “mathematically guaranteed”
864 to be unique. For example, GUID (Globally Unique Identifier) is a unique identifier that is based on the
865 simple principle that the total number of unique keys (or) is so large that the possibility of the same
866 number being generated twice is virtually zero.

867 This unique ID attribute SHOULD be used to uniquely identify collections of data as in the example below:
868 *Application A* supplies an xNL fragment containing some *PersonName* to *Application B*. The fragment
869 contains attribute *ID* with some unique value.

```
870 <n:PartyName n:ID="52F89CC0-5C10-4423-B367-2E8C14453926">  
871   <n:PersonName>  
872     <n:NameElement>Max Voskob</n:NameElement>  
873   </n:PersonName>  
874   <n:OrganisationName>  
875     <n:NameElement>Khandallah Laundering Ltd.</n:NameElement>  
876   </n:OrganisationName>  
877 </n:PartyName>
```

878

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

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

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

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

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

887 3.12 Schema Conformance

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

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

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

894 3.13 Schema Customisation Guidelines

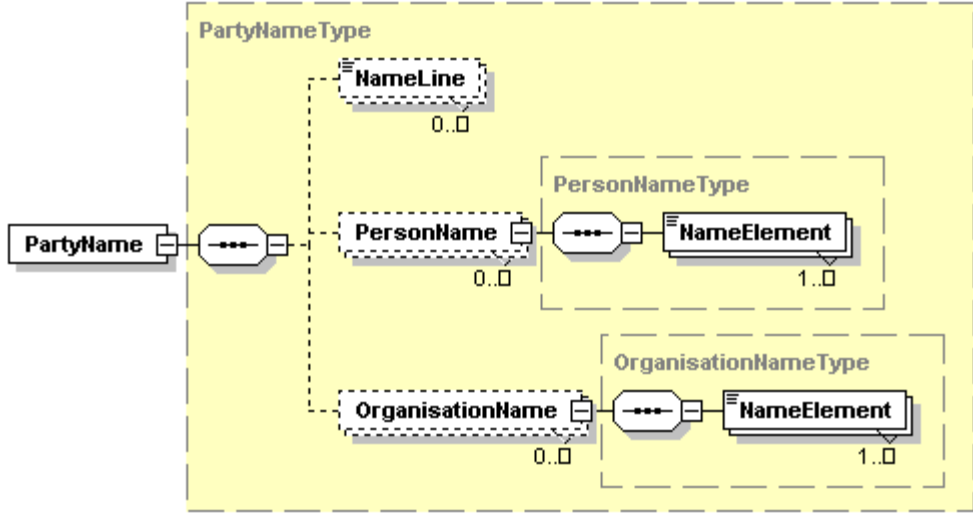
895 The broad nature and cultural diversity of entity “Name” makes it very difficult to produce one schema that
896 would satisfy all applications and all cultures while keeping the size and complexity of the schema
897 manageable. This specification allows some changes to the schema by adopters of the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

900 3.13.1 Namespace

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

903 3.13.2 Reducing the Entity Schema Structure

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



905 This structure allows for most names to be represented, with exception for
906
907 • organisation subdivision hierarchy (*SubdivisionName*), e.g. faculty / school / department

908 Any further reduction in structure MAY lead to loss of flexibility and expressive power of the schema.
909 Users MUST NOT remove any attributes from the schema. Attributes in the schema can be easily ignored
910 during the processing.

911 **3.13.2.1 Implications of changing Name Entity Schema**

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

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

917 **3.13.3 Customising the Code Lists/Enumerations of Name**

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

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

923 **NOTE:** The code list/enumeration values for different enumeration lists that are
924 provided as part of the specification are not complete. They only provides some
925 sample values (and in most cases no values) and it is up to the end users to
926 customise them to meet their data exchange requirements if the default values are
927 incomplete, not appropriate or over kill

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

933 **3.13.4 Using the Methodology to customise Name Schema to meet** 934 **application specific requirements**

935 The other approach to customise the CIQ Name schema (includes other entity schemas namely Party
936 and Address) without touching it is by using CVA. In this approach, one can use Schematron patterns to
937 define assertion rules to customise the *xNL.xsd* schema without modifying it. For example, it is possible to
938 customise *xNL.xsd* schema to restrict the use of elements, the occurrence of elements, the use of
939 attributes, and the occurrence of attributes, making elements and attributes mandatory, etc.

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

- 943 • CIQ Specifications that are tailored indirectly with the help of business rules to meet specific
944 application requirements
- 945 • Applications that use the customised CIQ Specifications with the help of business rules are still
946 compliant with the CIQ Specifications.

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

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

952

953

954

955 3.13.4.1 Constraining Name Schema using CVA – An Example

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

```
959 <n:PersonName>  
960   <n:NameElement n:ElementType="FirstName">Paruvachi</n:NameElement>  
961   <n:NameElement n:ElementType="FirstName">Ram</n:NameElement>  
962   <n:NameElement n:ElementType="MiddleName">Kumar</n:NameElement>  
963   <n:NameElement n:ElementType="LastName">Venkatachalam</n:NameElement>  
964   <n:NameElement n:ElementType="LastName">Gounder</n:NameElement>  
965 </n:PersonName>
```

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

```
972 <rule context="n:PersonName[not(parent::n:PartyName)]">  
973   <assert test=count(n:NameElement [@n:ElementType='FirstName']=1"  
974     >Must have exactly one FirstName component</assert>  
975   <assert test=count(n:NameElement [@n:ElementType='LastName']=1"  
976     >Must have exactly one LastName component</assert>  
977 </rule>
```

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

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

983 4 Entity “Address” (extensible Address Language)

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

986 The design concepts for “Address” are similar to “Name”. Refer to section 2.5 Common Design Concepts
987 for more information.

988 4.1 Semantics of “Address”

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

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

991 4.1.1 Example – Minimal Semantics (Unstructured/Free Text Data)

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

```
995 <a:Address>  
996   <a:FreeTextAddress>  
997     <a:AddressLine>Substation C</a:AddressLine>  
998     <a:AddressLine >17 James Street</a:AddressLine >  
999     <a:AddressLine>SPRINGVALE VIC 3171</a:AddressLine>  
1000   </a:FreeTextAddress>  
1001 </a:Address>
```

1002 It is up to the receiving application to parse this address and map it to the target data structure. It is
1003 possible that some sort of parsing software or human involvement will be required to accomplish the task.
1004 Data represented in this free formatted text form is classified as “poor quality” as it is subject to different
1005 interpretations of the data and will cause interoperability problems.

1006 Many common applications fall under this category.

1007 4.1.2 Example – Partial Semantics (Partially Structured Data)


1008 Assume that the address was captured in some semi-structured form such as State, Suburb and Street.

```
1009 <a:Address>  
1010   <a:AdministrativeArea>  
1011     <a:Name>WA</a:Name>  
1012   </a:AdministrativeArea>  
1013   <a:Locality>  
1014     <a:Name>OCEAN REEF</a:Name>  
1015   </a:Locality>  
1016   <a:Thoroughfare>  
1017     <a:NameElement>16 Patterson Street</a:NameElement>  
1018   </a:Thoroughfare>  
1019 </a:Address>
```


1020 In this example, the free text information resides in containers that provide some semantic information on
1021 the content. E.g. State -> AdministrativeArea, Suburb -> Locality, Street -> Thoroughfare. At the same
1022 time, the Thoroughfare element contains street name and number in one line as free text, which MAY not
1023 be detailed enough for data structures where street name and number are separate fields.


1024 Many common applications fall under this category.


1025


AddressType 

Complex type that defines the structure of an address with 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 densely populated area (a place) such as town, village, suburb, etc. A locality composes of many individual addresses.
- Thoroughfare** 


Details of the Access route along which buildings/lot/land are located, such as street, road, channel, crescent, avenue, etc.
- Premises** 


Details of the Premises (could be building(s), site, location, property, premise, place) 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 (e.g. an apartment complex or shopping centre) or even a vacant land (e.g. LOT).
- 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/installation where all mails are delivered and the post man/delivery service picks up the mails and delivers it to the recipients through a delivery mode.
- GeoRSS** 

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.
- LocationByCoordinates** 

Simple Geo-coordinates of the address/location

1027 4.1.3 Example – Full Semantics (Fully Structured Data)

1028 The following example illustrates an address structure that was decomposed into its atomic elements:

```
1029 <a:Address>
1030   <a:AdministrativeArea a:Type="state">
1031     <a:NameElement a:Abbreviation="true" a:NameType="Name">VIC</a:NameElement>
1032   </a:AdministrativeArea>
1033   <a:Locality a:Type="suburb">
1034     <a:NameElement a:NameType="Name">CLAYTON</a:NameElement>
1035     <a:SubLocality a:Type="Area">
1036       <a:NameElement a:NameType="Name">Technology Park</a:NameElement>
1037     </a:SubLocality>
1038   </a:Locality>
1039   <a:Thoroughfare a:Type="ROAD">
1040     <a:NameElement a:NameType="NameandType">Dandenong Road</a:NameElement>
1041     <a:Number a:IdentifierType="RangeFrom">200</a:Number>
1042     <a:Number a:IdentifierType="Separator">-</a:Number>
1043     <a:Number a:IdentifierType="RangeTo">350</a:Number>
1044     <a:SubThoroughfare a:Type="AVENUE">
1045       <a:NameElement a:NameType="NameAndType">Fifth Avenue</a:NameElement>
1046     </a:SubThoroughfare>
1047   </a:Thoroughfare>
1048   <a:Premises a:Type="Building">
1049     <a:NameElement a:NameType="Name">Toshiba Building</a:NameElement>
1050   </a:Premises>
1051   <a:PostCode>
1052     <a:Identifier>3168</a:Identifier>
1053   </a:PostCode>
1054 </a:Address>
```

1055 Few applications and in particular, applications dealing with data quality and integrity, fall under this
1056 category and the quality of data processed by these applications are generally high.

1057 4.2 Data Types

1058 All elements and attributes in *xAL* schema have strong data types.

1059 All free-text values of elements (text nodes) and attributes are constrained by a simple type
1060 “*NormalizedString*” (collapsed white spaces) defined in *CommonTypes.xsd*. Other XML Schema data
1061 types are also used throughout the schema.

1062 Other XML Schema defined data types (e.g. int, string, DateTime) are also used throughout *xAL*
1063 namespace.

1064 4.3 Code Lists (Enumerations)

1065 Use of code lists/enumerations is identical to use of code lists/enumerations for entity “*Name*”. Refer to
1066 section 3.3 for more information.

1067 Code Lists used in *xAL* for Option 1 reside in an “include” file *xAL-types.xsd* and for option 2 as separate
1068 genericode files.

1069 **NOTE:** The code list values for different code lists that are provided as part of
1070 the specifications are not complete. They only provides some sample values (and in
1071 most cases no values) and it is up to the end users to customise them to meet
1072 their data exchange requirements if the default values are incomplete, not
1073 appropriate or an over kill

1074 4.4 Order of Elements and Presentation

1075 Order of address elements **MUST** be preserved for correct presentation in a fashion similar to what is
1076 described in section 3.6.

1077

1078 **4.5 Data Mapping**

1079 Mapping data between *xAL* schema and a database is similar to that of entity “*Name*” as described in
1080 section 3.7.

1081 **4.5.1 Example – Normal Order**

1082 23 Archer Street
1083 Chatswood, NSW 2067
1084 Australia

1085 could be presented as follows

1086 <a:Address>
1087 <a:FreeTextAddress>
1088 <a:AddressLine>23 Archer Street</a:AddressLine>
1089 <a:AddressLine>Chatswood, NSW 2067</a:AddressLine>
1090 <a:AddressLine>Australia</a:AddressLine>
1091 </a:FreeTextAddress>
1092 </a:Address>

1093 and restored back to

1094 23 Archer Street
1095 Chatswood, NSW 2067
1096 Australia

1097 during data formatting exercise.

1098 Any other order of *AddressLine* tags in the XML fragment could lead to an incorrect presentation of the
1099 address.

1100 **4.6 Data Quality**

1101 *xAL* schema allows for data quality information to be provided as part of the entity using attribute
1102 *DataQuality* as for entity “*Name*”. Refer to section 3.8 for more information.

1103 **4.7 Extensibility**

1104 All elements in *Address* namespace are extensible as described in section 3.9.

1105 **4.8 Linking and Referencing**

1106 All linking and referencing rules described in section 3.10 apply to entity “*Address*”.

1107 **4.9 ID Attribute**

1108 Use of attribute ID is described in section 3.11.

1109 **4.10 Schema Conformance**

1110 Schema conformance described in section 3.12 is fully applicable to entity “*Address*”.

1111
1112
1113
1114
1115

1116 4.11 Address/Location Referenced By GeoRSS and Coordinates

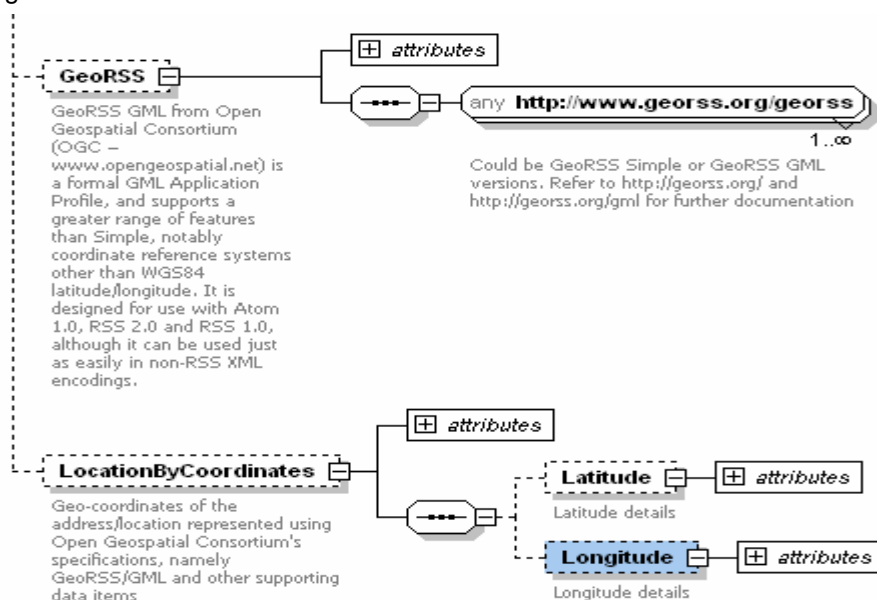
1117 xAL supports representation of Address/location in two ways namely,

- 1118 1. By using explicit coordinates with qualifiers for accuracy and precision, and
- 1119 2. By using the GeoRSS application profile, which expresses decimal degrees coordinates with
1120 accuracy and precision, and is implemented via external namespaces (either ATOM or RSS).

1121 Explicit coordinates are typically available from the process of geo-coding the street addresses.
1122 Coordinates are expressed in the *Latitude* and *Longitude* elements, including *DegreesMeasure*,
1123 *MinutesMeasure*, *SecondsMeasure*, and *Direction*. Data quality is expressed as attributes of coordinates
1124 including *Meridian*, *Datum* and *Projection*.

1125 GeoRSS incorporates a huge body of knowledge and expertise in geographical systems interoperability
1126 that can be reused for our purpose rather than re-inventing what has already been developed. The basic
1127 expression of *a:LocationByCoordinate* element in *xAL.xsd* schema has limits in utility for e-commerce
1128 applications. More interoperable expression of coordinate is possible via GeoRSS, due to the ability to
1129 reduce ambiguity introduced by requirements for different coordinate systems, units and measurements,
1130 or the ability to define more complex (non-point) geographic features.

1131 Support for GeoRSS and Location Coordinates for address/locations in *xAL.xsd* schema is shown in the
1132 following figure.



1133

1134 4.11.1 Using GeoRSS in xAL Schema

1135 As RSS becomes more and more prevalent as a way to publish and share information, it becomes
1136 increasingly important that location is described in an interoperable manner so that applications can
1137 **request, aggregate, share** and **map** geographically tagged feeds.

1138 GeoRSS (Geographically Encoded Objects for RSS feeds) enables geo-enabling, or tagging, "really
1139 simple syndication" (RSS) feeds with location information. GeoRSS proposes a standardised way in

1140 which location is encoded with enough simplicity and descriptive power to satisfy most needs to describe
1141 the location of Web content. GeoRSS MAY not work for every use, but it should serve as an easy-to-use
1142 geo-tagging encoding that is brief and simple with useful defaults but extensible and upwardly-compatible
1143 with more sophisticated encoding standards such as the OGC (Open Geospatial Consortium) GML
1144 (Geography Markup Language).

1145 GeoRSS was developed as a collaborative effort of numerous individuals with expertise in geospatial
1146 interoperability, RSS, and standards, including participants in the -- the W3C (World Wide Web
1147 Consortium)¹ and OGC (Open Geospatial Consortium)².

1148 GeoRSS is a formal GML Application Profile, with two flavours: 'GeoRSS Simple', which describes a
1149 point, and 'GeoRSS GML', which describes four essential types of shapes for geo-referencing (point, line,
1150 box and polygon).

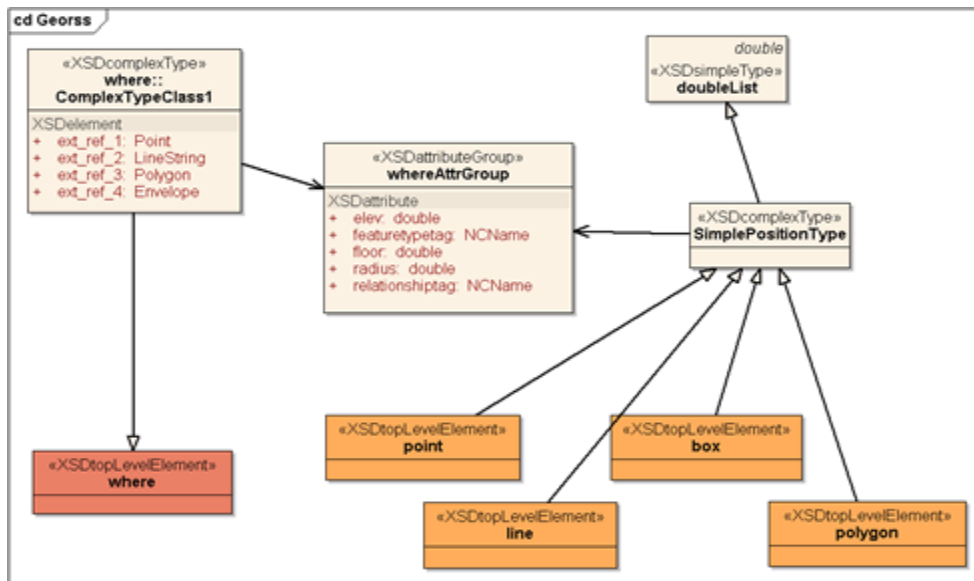
1151 GeoRSS Simple has greater brevity, but also has limited extensibility. When describing a point or
1152 coordinate, GeoRSS Simple can be used in all the same ways and places as GeoRSS GML.

1153 GeoRSS GML supports a greater range of features, notably coordinate reference systems other than
1154 WGS84 latitude/longitude. It is designed for use with Atom 1.0, RSS 2.0 and RSS 1.0, although it can be
1155 used just as easily in non-RSS XML encodings.

1156 Further detailed documentation and sample xml implementation information are published on the sites
1157 listed below:

- 1158 • <http://georss.org/>
- 1159 • <http://georss.org/gml>
- 1160 • <http://georss.org/atom>

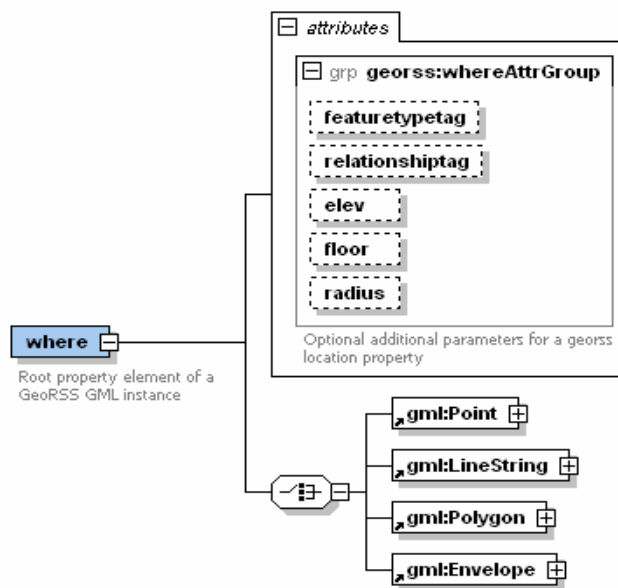
1161 The UML model for the GeoRSS application schema and the XML schema is shown below:



1162

¹ OGC – www.opengeospatial.net

² W3C – www.w3c.org



1163
 1164 GeoRSS is supported by an element `a:GeoRSS` in `xAL.xsd` schema as a non target namespace. The
 1165 content of `a:GeoRSS` must comply with the following requirements:

- 1166 • Be from the GeoRSS/GML/Atom namespace
- 1167 • Refer to finest level of address details available in the address structure that `a:GeoRSS` belongs to
- 1168 • Be used unambiguously so that there is no confusion whether the coordinates belong to the postal
 1169 delivery point (e.g. Post Box) or a physical address (e.g. flat) as it is possible to have both in the
 1170 same address structure.

1171 There is no restriction on the shape of the area, `a:GeoRSS` can describe be it a point, linear feature,
 1172 polygon or a rectangle.

1173 4.11.1.1 GeoRSS - Example

1174 The following are GeoRSS examples and demonstrate what GeoRSS Simple and GeoRSS GML
 1175 encodings look like. The location being specified is city center Ft. Collins.

1176 Simple GeoRSS:

```
1177 <georss:point>40.533203 -105.0712</georss:point>
```

1179 GML GeoRSS:

```
1180 <GeoRSS:where>
1181   <gml:Point>
1182     <gml:pos>40.533203 -105.0712</gml:pos>
1183   </gml:Point>
1184 </GeoRSS:where>
```

1185 These examples are in XML. However, RSS and GeoRSS are general models that can also be
 1186 expressed in other serializations such as Java, RDF or XHTML.

1187
 1188

1189 4.11.1.2 GeorRSS GML – Example

1190 A good way to describe a trip that has many places of interest like a boat trip or a hike is to specify the
1191 overall trip's path with a line as a child of the <feed>. Then mark each location of interest with a point in
1192 the <entry>.

```
1193 <feed xmlns="http://www.w3.org/2005/Atom"  
1194       xmlns:georss="http://www.georss.org/georss"  
1195       xmlns:gml="http://www.opengis.net/gml">  
1196   <title>Dino's Mt. Washington trip</title>  
1197   <link href="http://www.myisp.com/dbv/" />  
1198   <updated>2005-12-13T18:30:02Z</updated>  
1199  
1200   <author>  
1201     <name>Dino Bravo</name>  
1202     <email>dbv@example.org</email>  
1203   </author>  
1204  
1205   <id>http://www.myisp.com/dbv/</id>  
1206  
1207   <georss:where>  
1208     <gml:LineString>  
1209       <gml:posList>  
1210         45.256 -110.45 46.46 -109.48 43.84 -109.86 45.8 -109.2  
1211       </gml:posList>  
1212     </gml:LineString>  
1213   </georss:where>  
1214  
1215   <entry>  
1216     <title>Setting off</title>  
1217     <link href="http://www.myisp.com/dbv/1" />  
1218     <id>http://www.myisp.com/dbv/1</id>  
1219     <updated>2005-08-17T07:02:32Z</updated>  
1220     <content>getting ready to take the mountain!</content>  
1221     <georss:where>  
1222       <gml:Point>  
1223         <gml:pos>45.256 -110.45</gml:pos>  
1224       </gml:Point>  
1225     </georss:where>  
1226   </entry>  
1227  
1228   <entry>  
1229     <title>Crossing Muddy Creek</title>  
1230     <link href="http://www.myisp.com/dbv/2" />  
1231     <id>http://www.myisp.com/dbv/2</id>  
1232     <updated>2005-08-15T07:02:32Z</updated>  
1233     <content>Check out the salamanders here</content>  
1234     <georss:where>  
1235       <gml:Point>  
1236         <gml:pos>45.94 -74.377</gml:pos>  
1237       </gml:Point>  
1238     </georss:where>  
1239   </entry>  
1240 </feed>
```

1241 4.11.2 Defining Location Coordinates in xAL Schema

1242 If end users feel that GeorRSS GML is “overkill” or complex for their requirement and instead, want to just
1243 define the coordinates for location/address, *xAL.xsd* schema provides a default set of basic and
1244 commonly used elements representing explicit location coordinates through the element
1245 *a:LocationByCoordinates*.

1246 *a:LocationByCoordinates* element provides attributes namely, *Datum*, type of code used for Datum,
1247 *Meridian*, type of code used for Meridian, *Projection* and type of code used for Projection.

1248 *a:LocationByCoordinates/a:Latitude* and *a:LocationByCoordinates/a:Longitude* elements provide
1249 attributes namely, *DegreesMeasure*, *MinutesMeasure*, *SecondsMeasure*, and *Direction*.

1250

1251 4.12 Schema Customisation Guidelines

1252 Schema customisation rules and concepts described in section 3.13 are fully applicable to entity
1253 "Address".

1254 4.12.1 Customising the Code Lists/Enumerations of Address

1255 Addressing the 240+ country address semantics in one schema and at the same time keeping the
1256 schema simple is not trivial. Some countries have a city and some do not, some countries have counties,
1257 provinces or villages and some do not, some countries use canal names to represent the property on the
1258 banks of the canal, and, some countries have postal codes and some do not.

1259 Key components of international addresses that vary from country to country are represented in the
1260 specification using the schema elements namely, *Administrative Area*, *Sub Administrative Area*, *Locality*,
1261 *Sub Locality*, *Premises*, *Sub Premises*, *Thoroughfare*, and *Postal Delivery Point*. CIQ TC chose these
1262 names because they are independent of any country specific semantic terms such as City, Town, State,
1263 Street, etc. Providing valid and meaningful list of code lists/enumerations as default values to these
1264 elements that covers all countries is not a trivial exercise and therefore, this exercise was not conducted
1265 by CIQ TC. Instead, these elements are customisable using code lists/enumerations by end users to
1266 preserve the address semantics of each country which assists in improving the semantic quality of the
1267 address. To enable end users to preserve the meaning of the address semantics, the specification
1268 provides the ability to customise the schema using code lists/enumerations without changing the structure
1269 of the schema itself.

1270 For example, "State" defined in the code list/enumeration list for Administrative Area type could be valid
1271 for countries like India, Malaysia and Australia, but not for Singapore as it does not have the concept of
1272 "State". A value "Nagar" in the code list/enumeration list for Sub Locality type could be only valid for
1273 countries like India and Pakistan.

1274 If there is no intent to use the code list/enumeration list for the above schema elements, the code
1275 list/enumeration list can be ignored. There is requirement that the default values for the enumeration lists
1276 provided by the specification must exist. The list can be empty also. As long as the code list/enumeration
1277 list values are agreed between the parties involved in data exchange (whether data exchange between
1278 internal business system or with external systems), interoperability is not an issue.

1279 In Option 1 of representing code lists, the values clarifying the meaning of geographical entity types (e.g.
1280 *AdministrativeAreaType*, *LocalityAreaType*) in *xAL.xsd* were intentionally taken out of the main schema
1281 file into an "include" file (*xAL-types.xsd*) to make customisation easier. In Option 2 of Code List
1282 representation, these code lists are represented as separate .gc file in genericcode format.

1283 The values of the code lists/enumerations can be changed or new ones added as required.

1284 **NOTE:** The code list/enumeration values for different code/enumeration lists that are
1285 provided as part of the specifications are not complete. They only provide sample
1286 values (and in most case no values) and it is up to the end users to customise them to
1287 meet their data exchange requirements if the default values are incomplete, not
1288 appropriate or over kill

1289 4.12.1.1 End User Customised Code List - An Example

1290 In the example below, we use the country, Singapore. The default values provided by *xAL.xsd* for
1291 *AdministrativeAreaType* enumeration are given below. The user might want to restrict the values to meet
1292 only the address requirements for Singapore. Singapore does not have any administrative areas as it
1293 does not have state, city, or districts or provinces. So, the user can customise the schema by making the
1294 *AdministrativeAreaType* enumeration as an empty list as shown in the table below.

1295

1296

1297

1298

Original values for “AdministrativeAreaType” Code List	Customised Values
City	
State	
Territory	
Province	

1299 This level of flexibility allows some customisation of the schema through changing the enumerations only,
1300 without changing the basic structure of the schema. It is important to ensure that all schema users
1301 involved in data exchange use the same enumerations for interoperability to be successful. This has to be
1302 negotiated between the data exchange parties and a proper governance process SHOULD be in place to
1303 manage this process.

1304 4.12.1.2 Implications of changing Address Entity Schema

1305 Any changes to the Address Entity schema (*xAL.xsd*) are likely to break the compatibility one way or
1306 another.

1307 It MAY be possible that an XML fragment created for the original schema is invalid for the altered schema
1308 or vice versa. This issue needs to be considered before making any changes to the schema that could
1309 break the compatibility.

1310 4.12.2 Using CVA to customise CIQ Address Schema to meet application 1311 specific requirements

1312 The other approach to customise the CIQ address schema (*xAL.xsd*) without modifying it is by CVA. In
1313 this approach, one can use Schematron patterns to define assertion rules to customise CIQ address
1314 schema without modifying it. For example, it is possible to customise CIQ address schema to restrict the
1315 use of address entities that are not required for a specific country. For example, a country like Singapore
1316 will not need address entities namely, *Administrative Area*, *Sub Administrative Area*, *Sub Locality*, *Rural*
1317 *Delivery* and *Post Office*. These entities can be restricted using Schematron based assertion rules.
1318 Some might want to just use free text address lines and a few of the address entities like locality and
1319 postcode. Schematron assertion rules help users to achieve this.

1320 **NOTE:** The business rules used to constraint CIQ address schema SHOULD be agreed by
1321 all the parties that are involved in data exchange of CIQ based address data to
1322 ensure interoperability and the rules SHOULD be governed.

1323 4.12.2.1 Constraining CIQ Address Schema using CVA – Example 1

1324 Let us use the country “Singapore” as an example again. Let us say that the country “Singapore” only
1325 requires the following address entities defined in *xAL.xsd* and does not require the rest of the entities as
1326 they are not applicable to the country:

- 1327 • Country
- 1328 • Locality
- 1329 • Thoroughfare
- 1330 • PostCode

1331
1332
1333
1334
1335

1336 This restriction can be achieved without modifying the *xAL.xsd* schema and by applying the following
1337 schematron pattern rules outside of *xAL.xsd* schema as follows:

```
1338 <rule context="a:Address/*">  
1339   <assert test="(name()='a:Country') or (name()='a:PostCode') or  
1340     (name()='a:Thoroughfare') or (name()='a:Locality')"  
1341     >Invalid data element present in the document  
1342   </assert>  
1343 </rule>
```

1344 The above simple rule restricts the use of other elements and attributes in *xAL.xsd* when an XML
1345 instance document is produced and validated.

1346 Now let us take the following XML instance document:

```
1347 <a:Address>  
1348   <a:Country>  
1349     <a:NameElement>Singapore</a:NameElement>  
1350   </a:Country>  
1351   <a:AdministrativeArea>  
1352     <a:NameElement></a:NameElement>  
1353   </a:AdministrativeArea>  
1354   <a:Locality>  
1355     <a:NameElement>NUS Campus</a:NameElement>  
1356   </a:Locality>  
1357   <a:Thoroughfare>  
1358     <a:NameElement>23 Woodside Road</a:NameElement>  
1359   </a:Thoroughfare>  
1360   <a:Premises>  
1361     <a:NameElement></a:NameElement>  
1362   </a:Premises>  
1363   <a:PostCode>  
1364     <a:Identifier>51120</a:Identifier>  
1365   </a:PostCode>  
1366 </a:Address>  
1367
```

1368 When the above document instance is validated using CVA, pass one validation (structure validation
1369 against *xAL.xsd*) will be successful. Pass two validation (business rules and value validation) will report
1370 the following errors:

```
1371 Invalid data element present in the document  
1372   :/a:Address/a:AdministrativeArea  
1373 Invalid data element present in the document  
1374   :/a:Address/a:Premises
```

1375 4.12.2.2 Constraining CIQ Address Schema using CVA – Example 2

1376 Let us consider another example where an application requires using only the free text address lines in
1377 *xAL.xsd* and no other address entities.

1378 This restriction can be achieved without modifying the *xAL.xsd* schema and by applying the following
1379 schematron pattern rules outside of the schema as follows:

```
1380 <rule context="a:Address/*">  
1381   <assert test="name()='a:FreeTextAddress'">  
1382     >Invalid data element present in the document  
1383   </assert>  
1384 </rule>
```

1385 The above simple rule restricts the use of elements and attributes other than "*FreeTextAddress*" element
1386 in *xAL.xsd* when an XML instance document is produced and validated.

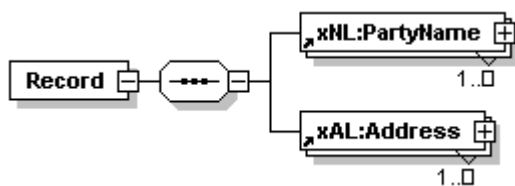
1387 5 Combination of “Name” and “Address” (extensible 1388 Name and Address Language)

1389 *xNAL* (*Name* and *Address*) schema is a container for combining related names and addresses. This
1390 specification recognises two ways of achieving this and they are:

- 1391 • Binding multiple names to multiple addresses (element *xnal:Record*)
- 1392 • Binding multiple names to a single address for postal purposes (element *xnal:PostalLabel*)

1393 5.1 Use of element *xnal:Record*

1394 Element *xnal:Record* is a binding container that shows that some names relate to some addresses as in
1395 the following diagram:



1396 The relationship type is application specific, but in general it is assumed that a person defined in the *xNL*
1397 part have some connection/link with an address specified in the *xAL* part. Use attributes from other
1398 namespaces to specify the type of relationships and roles of names and addresses.
1399

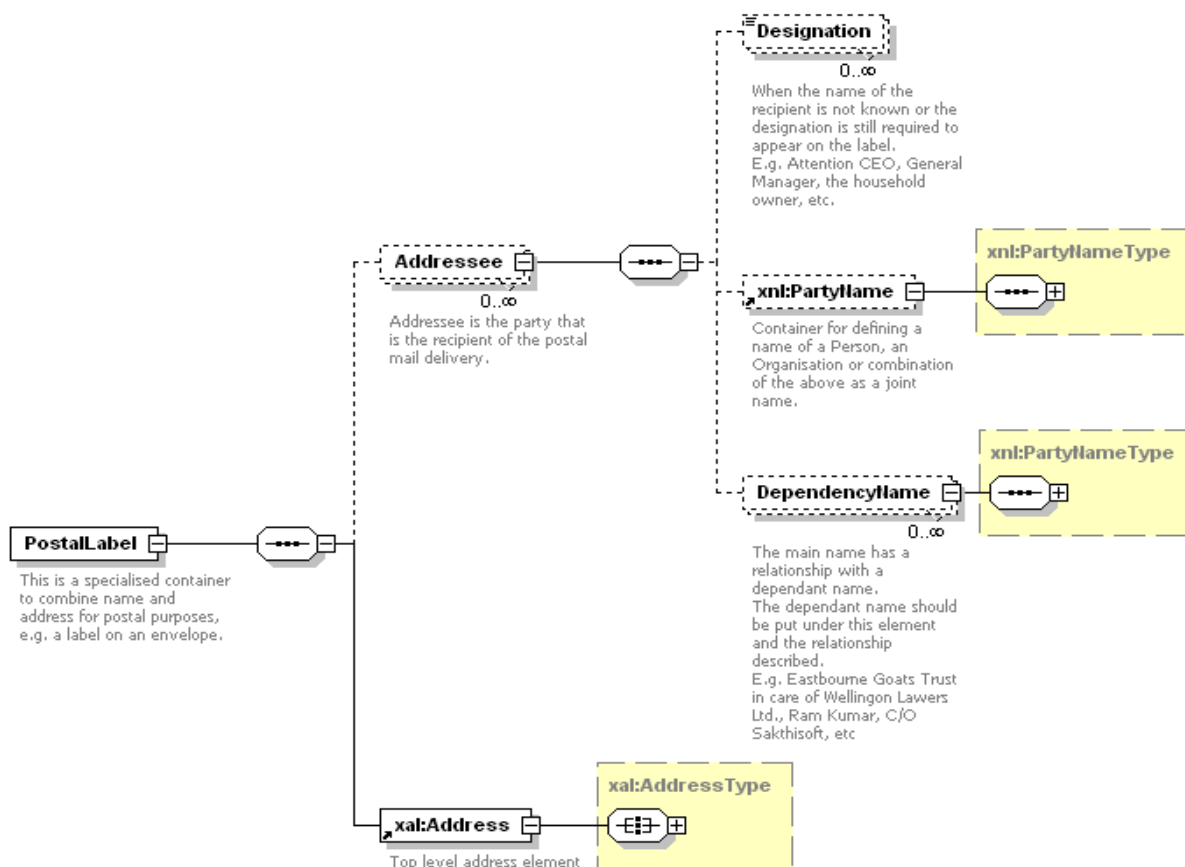
1400 5.1.1 Example

```
1401 Mr H G Guy, 9 Uxbridge Street, Redwood, Christchurch 8005
1402 <xnal:Record>
1403   <n:PartyName>
1404     <n:NameLine>Mr H G Guy</n:NameLine>
1405   </n:PartyName>
1406   <a:Address>
1407     <a:Locality>
1408       <a:Name>Christchurch</a:Name>
1409       <a:SubLocality>Redwood</a:SubLocality>
1410     </a:Locality>
1411     <a:Thoroughfare>
1412       <a:Number>9</a:Number>
1413       <a:NameElement>Uxbridge Street</a:NameElement>
1414     </a:Thoroughfare>
1415     <a:PostCode>
1416       <a:Identifier>8005</a:Identifier>
1417     </a:PostCode>
1418   </a:Address>
1419 </xnal:Record>
```

1420
1421
1422
1423
1424

1425 **5.2 Use of element xnal:PostalLabel**

1426 Element *xnal:PostalLabel* is a binding container that provides elements and attributes for information
 1427 often used for postal / delivery purposes, as in the following diagram. This has two main containers, an
 1428 addressee and the address:



1429 This structure allows for any number of recipients to be linked to a single address with some delivery
 1430 specific elements such as *Designation* and *DependencyName*.
 1431

1432 **5.2.1 Example**

```

1433 Attention: Mr S Mart
1434 Director
1435 Name Plate Engravers
1436 The Emporium
1437 855 Atawhai Drive
1438 Atawhai
1439 Nelson 7001
  
```

1440 translates into the following *xNAL* fragment:

```

1441 <xnal:PostalLabel>
1442   <xnal:Addressee>
1443     <xnal:Designation>Attention: Mr S Mart</xnal:Designation>
1444     <xnal:Designation>Director</xnal:Designation>
1445     <n:PartyName>
1446       <n:NameLine>Name Plate Engravers</n:NameLine>
1447     </n:PartyName>
1448   </xnal:Addressee>
1449   <a:Address>
1450     <a:Locality>
1451       <a:Name>Nelson</a:Name>
1452       <a:SubLocality>Atawhai</a:SubLocality>
  
```

1453
1454
1455
1456
1457
1458
1459
1460
1461
1462

```

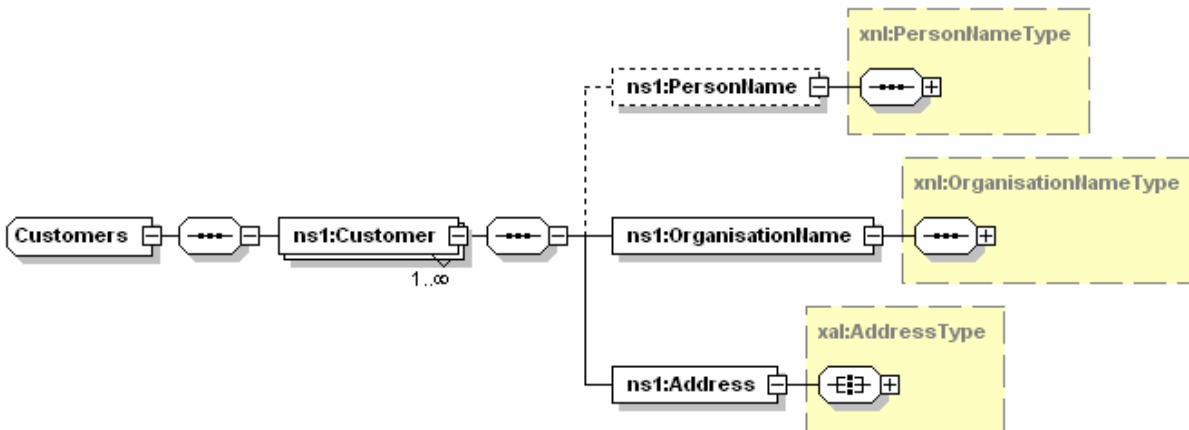
</a:Locality>
<a:Thoroughfare>
  <a:NameElement>Atawhai Drive</a:NameElement>
  <a:Number>855</a:Number>
</a:Thoroughfare>
<a:PostCode>
  <a:Identifier>7001</a:Identifier>
</a:PostCode>
</a:Address>
</xnal:PostalLabel>

```

1463 **5.3 Creating your own Name and Address Application Schema**

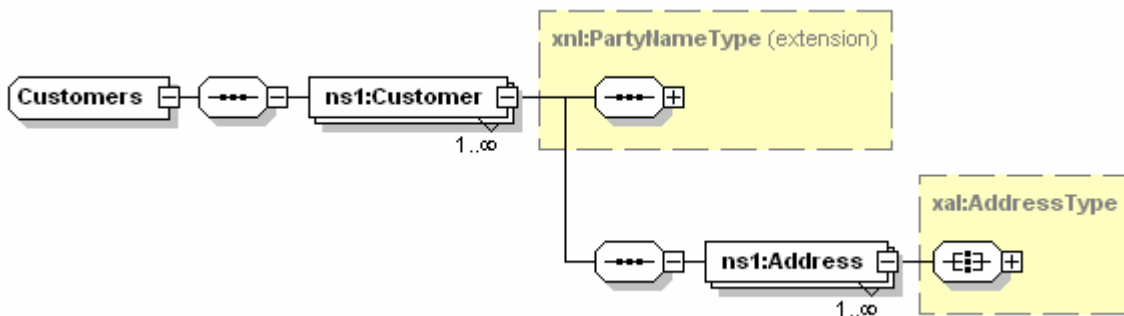
1464 Users can use the *xNL* and *xAL* constructs and create their own name and address container schema to
 1465 meet their specific requirements rather than using a container element called "Record" as in *xNAL* if they
 1466 believe that *xNAL* schema does not meet their requirements. This is where the power of CIQ
 1467 Specifications comes in to play. It provides the basic party constructs to enable users to reuse the base
 1468 constructs of CIQ specifications as part of their application specific data model and at the same time
 1469 meeting their application specific requirements.

1470 For example, users can create a schema called *Customers.xsd* that could reuse *xNL* and *xAL* to
 1471 represent their customers. This is shown in the following figure:



1472 In the above figure, *PersonName* is OPTIONAL.

1474



1475 In the above figure, "Customer" is of type "Party" as defined in *xNL* schema. "Customer" is then extended
 1476 to include "Address" element that is of type "Address" as defined in *xAL* schema.

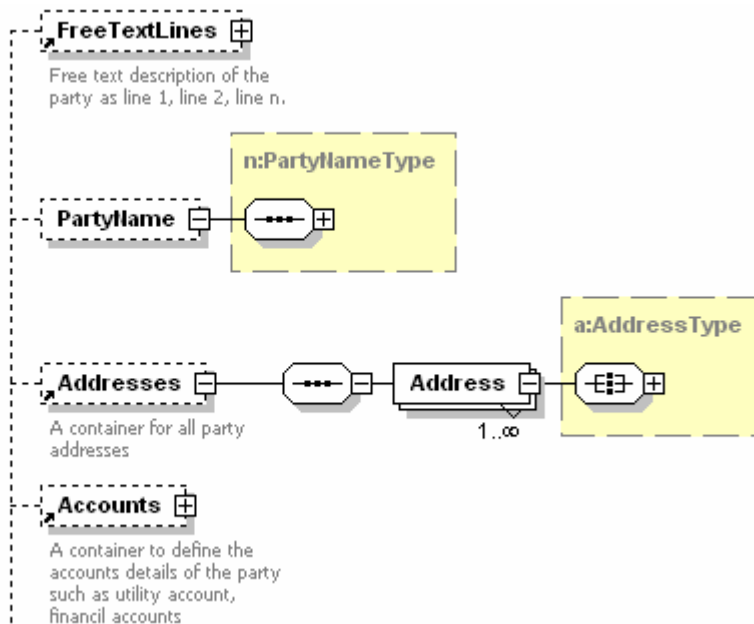
1478

6 Entity “Party” (extensible Party Information Language)

1479
1480
1481 Entity “Party” encapsulates some most commonly used unique characteristics/attributes of *Person* or
1482 *Organisation*, such as name, address, personal details, contact details, physical features, etc.
1483 This assists in uniquely identifying a party with these unique party attributes.
1484 The schema consists of top level containers that MAY appear in any order or MAY be omitted. The
1485 containers are declared globally and can be reused by other schemas. The full schema for defining a
1486 *Party* can be found in *xPIL.xsd* file with enumerations in *xPIL-types.xsd* file for Code List Option 1 and .gc
1487 files for Code List Option 2. See the sample XML files for examples.
1488 *xPIL* provides a number of elements/attributes that are common to both a person and an organisation
1489 (e.g. account, electronic address identifier, name, address, contact numbers, membership, vehicle, etc).
1490 *xPIL* provides a number of elements/attributes that are applicable to a person only (e.g. gender, marital
1491 status, age, ethnicity, physical information, hobbies, etc)
1492 *xPIL* provides a number of elements/attributes that are applicable to an organisation only (e.g. industry
1493 type, registration details, number of employees, etc)

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

1494
1495
1496 “Name” of *xPIL* schema reuses *PartyNameType* constructs from *xNL* namespace and “Address” of the
1497 *xPIL* schema reuses *AddressType* construct from *xAL* namespace as illustrated in the following diagram:



1498
1499
1500 The design paradigm for this *xPIL* schema is similar to those of Name and Address entities. Likewise, it is
1501 possible to combine information at different detail and semantic levels.
1502
1503

1504 6.2 Party Structures - Examples

1505 The following examples illustrate use of a selection of party constructs.

1506 6.2.1 Example – Qualification Details

```
1507 <p:Qualifications>
1508   <p:Qualification>
1509     <p:QualificationElement
1510     p:Type="QualificationName">BComp.Sc.</p:QualificationElement>
1511     <p:QualificationElement
1512     p:Type="MajorSubject">Mathematics</p:QualificationElement>
1513     <p:QualificationElement
1514     p:Type="MinorSubject">Statistics</p:QualificationElement>
1515     <p:QualificationElement p:Type="Award">Honours</p:QualificationElement>
1516     <p:InstitutionName>
1517       <n:NameLine>University of Technology Sydney</n:NameLine>
1518     </p:InstitutionName>
1519   </p:Qualification>
1520 </p:Qualifications>
```

1521 6.2.2 Example – Birth Details

```
1522 <p:BirthInfo p:BirthDateTime="1977-01-22T00:00:00"/>
```

1523 6.2.3 Example – Driver License

```
1524 <p:Document p:ValidTo="2004-04-22T00:00:00">
1525   <p:IssuePlace>
1526     <a:Country>
1527       <a:Name>Australia</a:Name>
1528     </a:Country>
1529     <a:AdministrativeArea>
1530       <a:Name>NSW</a:Name>
1531     </a:AdministrativeArea>
1532   </p:IssuePlace>
1533   <p:DocumentElement p:Type="DocumentID">74183768C</p:DocumentElement>
1534   <p:DocumentElement p:Type="DocumentType">Driver License</p:DocumentElement>
1535   <p:DocumentElement p:Type="Privilege">Silver</p:DocumentElement>
1536   <p:DocumentElement p:Type="Restriction">Car</p:DocumentElement>
1537 </p:Document>
```

1538 6.2.4 Example – Contact Phone Number

```
1539 <p>ContactNumber p:MediaType="Telephone" p>ContactNature="Business Line"
1540 p>ContactHours="9:00AM - 5:00PM">
1541   <p>ContactNumberElement p:Type="CountryCode">61</p>ContactNumberElement>
1542   <p>ContactNumberElement p:Type="AreaCode">2</p>ContactNumberElement>
1543   <p>ContactNumberElement p:Type="LocalNumber">94338765</p>ContactNumberElement>
1544 </p>ContactNumber>
```

1545 6.2.5 Example – Electronic Address Identifiers

```
1546 <p:ElectronicAddressIdentifiers>
1547   <p:ElectronicAddressIdentifier p:Type="SKYPE" p:Usage="Personal">rkumar
1548 </p:ElectronicAddressIdentifiers>
1549   <p:ElectronicAddressIdentifier p:Type="EMAIL" p:Usage="Business">ram.kumar@email.com
1550 </p:ElectronicAddressIdentifiers>
1551   <p:ElectronicAddressIdentifier p:Type="URL"
1552   p:Usage="Personal">http://www.ramkumar.com
1553 </p:ElectronicAddressIdentifiers>
1554
```

1555

1556 **6.3 Dealing with Joint Party Names**

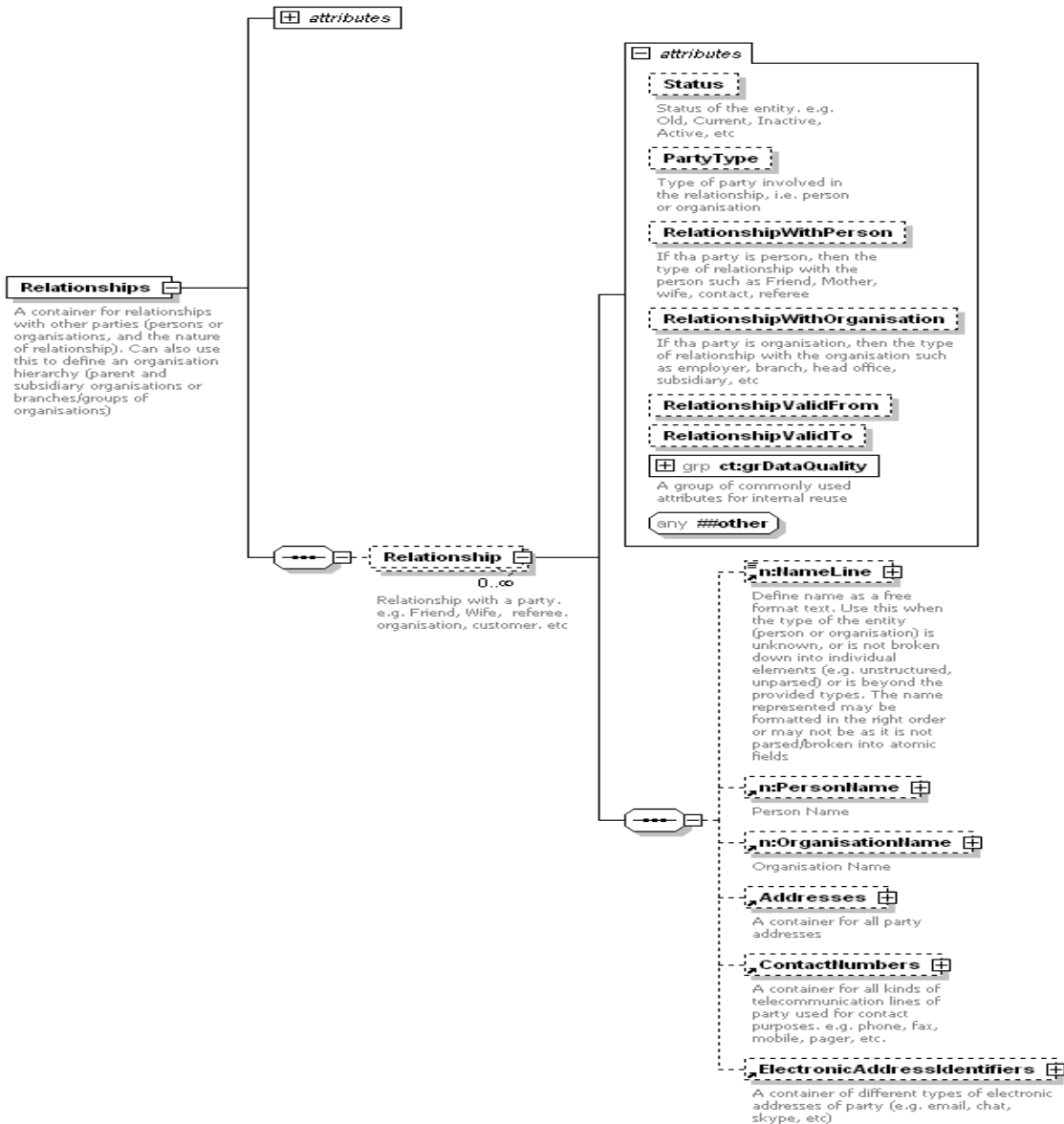
1557 *xPIL* schema represents details of a *Party*. The *Party* has a name as specified in *n:PartyName* element. A
1558 "Party" can be a unique name (e.g. A person or an Organisation) or a joint name (e.g. Mrs. Sarah
1559 Johnson and Mr. James Johnson (or) Mrs. & Mr. Johnson). In this case, all the other details of the party
1560 defined using *xPIL* apply to the party as a whole (i.e. to both the persons in the above example) and not
1561 to one of the Parties (e.g. say only to Mrs. Sarah Johnson or Mr. James Johnson in the example). Also,
1562 all the addresses specified in *Addresses* element relate to the *Party* as a whole (i.e. applies to both Mrs.
1563 and Mr. Johnson in this example).

1564 If for example, Mrs. Sarah Johnson and Mr. James Johnson have to be defined separately with their own
1565 unique characteristics (e.g. address, vehicle, etc), then each person SHOULD be defined as an individual
1566 party.

1567 **6.4 Representing Relationships with other Parties**

1568 *xPIL* provides the ability to also define simple one to one relationships between a party (person or an
1569 organisation) and other parties (person or organisation). This is shown in the following diagram (an
1570 extract of XML schema).

1571 However, it is strongly advised that users interested in implementing relationships between parties using
1572 CIQ specifications SHOULD use CIQ *xPRL* (*extensible Party Relationships Language*) specification
1573 version 3.0 exclusively defined for dealing with party relationships.



- 1574
- 1575 Examples of relationships include, Friend, Spouse, Referee, Contact, etc for a person, and Client,
- 1576 customer, branch, head office, etc for an organisation.
- 1577 Details of each party involved in the relationship can be defined namely, Person Name, Organisation
- 1578 Name, Contact Numbers and Electronic Address Identifiers.
- 1579 The "Relationship" element provides the relationship details between the parties. It's attribute *Status*
- 1580 defines the status of relationship; attribute *RelationshipWithPerson* defines the type of relationship with
- 1581 the person (e.g. friend, spouse) if the party is a person; attribute *RelationshipWithOrganisation* defines
- 1582 the type of relationship with the organisation (e.g. client, branch, subsidiary) if the party is an organisation;
- 1583 attributes *RelationshipValidFrom* and *RelationshipValidTo* defines the dates of the relationship with the
- 1584 party.
- 1585
- 1586
- 1587
- 1588

1589 **6.4.1 Example – Person Relationship with other Persons of type “Friend”**

```
1590 <p:Relationships>
1591   <p:Relationship p:RelationshipWithPersonGroup="Friend">
1592     <p:PartyDetails>
1593       <p:PersonName>
1594         <p:NameElement="FullName">Andy Chen</NameElement>
1595       </p:PersonName>
1596     </p:PartyDetails>
1597   </p:Relationship>
1598   <p:Relationship p:RelationshipWithPersonGroup="Friend">
1599     <p:PartyDetails>
1600       <p:PersonName>
1601         <p:NameElement="FullName">John Freedman</NameElement>
1602       </p:PersonName>
1603     </p:PartyDetails>
1604   </p:Relationship>
1605   <p:Relationship p:RelationshipWithPersonGroup="Friend">
1606     <p:PartyDetails>
1607       <p:PersonName>
1608         <p:NameElement="FullName">Peter Jackson</NameElement>
1609       </p:PersonName>
1610     </p:PartyDetails>
1611   </p:GroupRelationship>
1612 </p:Relationships>
```

1613 **6.4.2 Example – Organisation Relationship with other Organisations of type**
1614 **“Branch”**

```
1615 <p:Relationships>
1616   <p:Relationship p:PartyType="Organisation" p:RelationshipWithOrganisation="Branch">
1617     <p:NameLine>XYZ Pty. Ltd</p:NameLine>
1618     <p:Address>
1619       <p:FreeTextAddress>
1620         <p:AddressLine>23 Archer Street, Chastwood, NSW 2067,
1621           Australia
1622         </p:AddressLine>
1623       </p:FreeTextAddress>
1624     </p:Address>
1625   </p:Relationship>
1626   <p:Relationship p:PartyType="Organisation" p:RelationshipWithOrganisation="Branch">
1627     <p:NameLine>XYZ Pte. Ltd</p:NameLine>
1628     <p:Address>
1629       <p:FreeTextAddress>
1630         <p:AddressLine>15, Meena Rd, K.K.Nagar, Chennai 600078
1631           India
1632         </p:AddressLine>
1633       </p:FreeTextAddress>
1634     </p:Address>
1635   </p:Relationship>
1636 </p:Relationships>
1637
```

1638 **6.4.3 Example – Person Relationship with another Person**

```
1639 <p:Relationships>
1640   <p:Relationship p:RelationshipWithPersonGroup="Son">
1641     <p:PersonName>
1642       <p:NameElement="FullName">Andy Chen</NameElement>
1643     </p:PersonName>
1644   </p:Relationship>
1645 </p:Relationships>
```

1646

1647 6.5 Data Types

1648 All elements and attributes in *xPIL* schema have strong data types.

1649 All free-text values of elements (text nodes) and attributes are constrained by a simple type
1650 “*NormalizedString*” (collapsed white spaces) defined in *CommonTypes.xsd*. Other XML Schema data
1651 types are also used throughout the schema.

1652 Other XML Schema defined data types are also used throughout the schema.

1653 6.6 Code Lists (Enumerations)

1654 Use of code lists/enumerations is identical to use of code lists for entity “*Name*”. Refer to section 3.3 for
1655 more information.

1656 Code lists/enumerations used in *xPIL* for code list option 1 reside in an “include” *xPIL-types.xsd*. Code
1657 lists/enumerations used in *xPIL* for code list option 2 reside as .gc genericcode files.

1658 **NOTE:** The code list/enumeration values for different code lists/enumeration lists
1659 that are provided as part of the specifications are not complete. They only
1660 provides some sample values (and in most cases no values) and it is up to the end
1661 users to customise them to meet their data exchange requirements if the default
1662 values are incomplete, not appropriate or over kill

1663 6.7 Order of Elements and Presentation

1664 Order of elements without qualifier (@...type attribute) MUST be preserved for correct presentation as
1665 described in section 3.6.

1666 6.8 Data Mapping

1667 Mapping data between *xPIL* schema and a database is similar to that of entity “*Name*” as described in
1668 section 3.7.

1669 6.9 Data Quality

1670 *xPIL* schema allows for data quality information to be provided as part of the entity using attribute
1671 *DataQuality* as for entity “*Name*”. Refer to section 3.8 for more information.

1672 6.10 Extensibility

1673 All elements in *Party* namespaces are extensible as described in section 3.10.

1674 6.11 Linking and Referencing

1675 All linking and referencing rules described in section 3.9 apply to entity “*Party*”.

1676 The following example illustrates *PartyName* elements that reference other *PartyName* element that
1677 resides elsewhere, in this case outside of the document.

```
1678 <a:Contacts xmlns:a="urn:acme.org:corporate:contacts">  
1679   <xnl:PartyName xlink:href="http://example.org/party?id=123445"/>  
1680   <xnl:PartyName xlink:href="http://example.org/party?id=83453485"/>  
1681 </a:Contacts>
```

1682 This example presumes that the recipient of this XML fragment has access to resource
1683 “*http://example.org/party*” (possibly over HTTP/GET) and that the resource returns as *PartyName* element
1684 as an XML fragment of *text/xml* MIME type.

1685 Use of attribute ID is described in section 3.11.

1686 **6.12 Schema Conformance**

1687 Schema conformance described in section 3.12 is fully applicable to entity "Party".

1688 **6.13 Schema Customisation Guidelines**

1689 Schema customisation rules and concepts described in section 3.13 are fully applicable to entity "Party".

1690 **6.13.1 Customising the Code Lists/Enumerations of Party**

1691 If there is no intent to use the code list/enumeration list for the xPIL schema elements, the code
1692 list/enumeration list can be ignored. There is no absolute must rule that the default values for the
1693 enumeration lists provided by the specification must exist. The list can be empty also. As long as the code
1694 list/enumeration list values are agreed between the parties involved in data exchange (whether data
1695 exchange between internal business system or with external systems), interoperability is not an issue.

1696 In Option 1 of representing code lists, the values clarifying the meaning of party element types (e.g.
1697 *DocumentType, ElectronicAddressIdentifierType*) in *xPIL.xsd* were intentionally taken out of the main
1698 schema file into an "include" file (*xPIL-types.xsd*) to make customisation easier. In Option 2 of Code List
1699 representation, these code lists are represented as separate .gc file in genericcode format.

1700 The values of the code lists/enumerations can be changed or new ones added as required.

1701 **NOTE:** The code list/enumeration values for different code/enumeration lists that are
1702 provided as part of the specifications are not complete. They only provide sample
1703 values (and in most case no values) and it is up to the end users to customise them to
1704 meet their data exchange requirements if the default values are incomplete, not
1705 appropriate or over kill

1706 **6.13.1.1 End User Customised Code List - An Example**

1707 In the example below, we use *Identifier* element of *xPIL.xsd*. The default values provided by CIQ
1708 Specification for *Identifier* type's enumeration are given below. The user might want to restrict these
1709 values. So, the user can customise the code list for *Identifier* types by making the
1710 *PartyIdentifierTypeEnumeration* with the required values as shown in the table below.

Default values for "PartyIdentifierTypeList" Code List	Customised values
TaxID	TaxID
CompanyID	
NationalID	
RegistrationID	

1711 This level of flexibility allows some customisation of the schema through changing the code
1712 list/enumerations only, without changing the basic structure of the schema. It is important to ensure that
1713 all schema users involved in data exchange use the same cod list/enumerations for interoperability to be
1714 successful. This has to be negotiated between the data exchange parties and a proper governance
1715 process SHOULD be in place to manage this process.

1716 **6.13.1.2 Implications of changing Party Entity Schema**

1717 Any changes to the Party Entity schema (*xPIL.xsd*) are likely to break the compatibility one way or
1718 another.

1719 It MAY be possible that an XML fragment created for the original schema is invalid for the altered schema
1720 or vice versa. This issue needs to be considered before making any changes to the schema that could
1721 break the compatibility.

1722 **6.13.2 Using CVA to customise Party Schema to meet application specific**
1723 **requirements**

1724 The other approach to customise the CIQ party schema (*xPIL.xsd*) without touching it is by using CVA. In
1725 this approach, one can use Schematron patterns to define assertion rules to customise party schema
1726 without touching or modifying it. For example, it is possible to customise party schema to restrict the use
1727 of party entities (elements and attributes) that are not required for a specific application. These entities
1728 can be restricted using Schematron based assertion rules.

1729 **NOTE:** The business rules used to constraint CIQ party schema SHOULD be agreed by
1730 all the parties that are involved in data exchange of CIQ based party data to
1731 ensure interoperability and the rules SHOULD be governed.

1732

1733

1734

7 Differences between two types of Entity Schemas for CIQ Specifications

1735

1736

1737 CIQ Specifications comes with two types of entity schemas (*xNL.xsd*, *xAL.xsd*, *xPIL.xsd*, and *xNAL.xsd*)
1738 based on the type of code lists/enumerations used. The types of code lists/enumerations options used
1739 are:

1740 **Option1 (Default):** All code lists for an entity represented using XML schema (in one file) and “included”
1741 in the appropriate entity schema (*xNL-types.xsd*, *xAL-types.xsd*, *xNAL-types.xsd*, and *xPIL-types.xsd*).

1742

1743 **Option 2:** Code Lists represented using Genericode structure of OASIS Codelist TC. Each enumeration
1744 list in option 1 is a separate “.gc” file in this option.

7.1 Files for Option 1 (The Default)

1746 Following are the XML schema files provided as default in CIQ Specifications package for Option 1:

1747

- *xNL.xsd*

1748

- *xNL-types.xsd* (**13** Default Code Lists defined for *xNL*)

1749

- *xAL.xsd*

1750

- *xAL-types.xsd* (**32** Default Code Lists defined for *xAL*)

1751

- *xPIL.xsd*

1752

- *xPIL-types.xsd* (**60** Default Code Lists defined for *xPIL*)

1753

- *xNAL.xsd*

1754

- *xNAL-types.xsd* (**2** Default Code List defined for *xNAL*)

1755

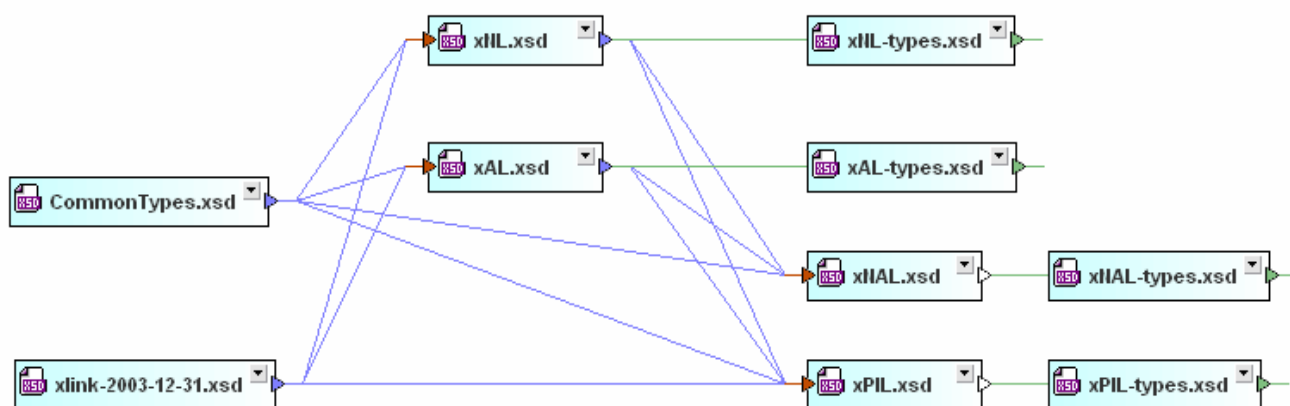
- *CommonTypes.xsd* (**2** Default Code Lists defined for Common Type for all entities)

1756

- *xlink-2003-12-21.xsd*

1757

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



1758

1759

1760

1761

1762 **7.2 Files for Option 2**

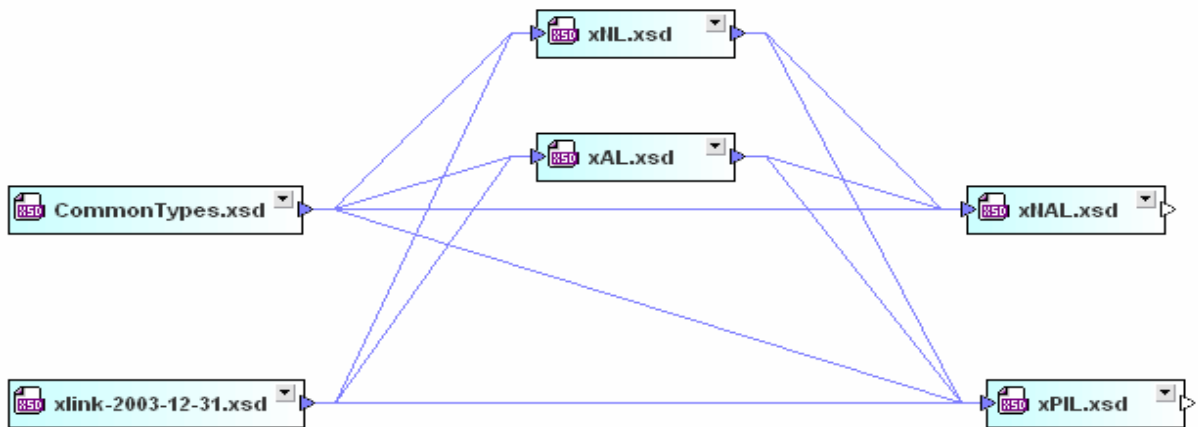
1763 Following are the files provided as default in CIQ Specifications package for Option 2:

1764 **7.2.1 XML Schema Files**

- 1765 • *xNL.xsd*
- 1766 • *xAL.xsd*
- 1767 • *xPIL.xsd*
- 1768 • *xNAL.xsd*
- 1769 • *CommonTypes.xsd*
- 1770 • *xlink-2003-12-21.xsd*

1771 No **-types.xsd* files exist in Option 2 as all the code lists are defined as genericcode files.

1772 The relationship between the different schemas for Option 2 is shown in the following figure. As you can
1773 see, the enumeration list XML schemas do not exist. Instead, each CIQ entity (Name, Address, and
1774 Party) has a set of genericcode based Code List files (.gc).



1775

1776 **7.2.2 Genericcode Based Code List Files**

1777 **7.2.2.1 For Name (xNL)**

1778 12 default genericcode based code list files with .gc extension. Each enumeration list in Option 1 is defined
1779 as a separate file in Option 2.

1780 **7.2.2.2 For Address (xAL)**

1781 32 default genericcode based code list files with .gc extension. Each enumeration list in Option 1 is defined
1782 as a separate file in Option 2.

1783 **7.2.2.3 For Name and Address (xNAL)**

1784 2 default genericcode based code list files with .gc extension. The enumeration list in Option 1 is defined
1785 as a separate file in Option 2.

1786 **7.2.2.4 For Party (xPIL)**

1787 54 default genericcode based code list files with .gc extension. Each enumeration list in Option 1 is defined
1788 as a separate file in Option 2.

1789

1790 **7.2.2.5 For Common Types**

1791 2 default genericcode based code list files with .gc extension.

1792 **7.3 Namespace Assignment**

1793 Both the types of entity schemas (for option 1 and option 2) use the same namespaces to ensure that the
1794 XML instance documents generated from any of these two options are compatible with both types of CIQ
1795 entity XML schemas.

1796 **7.4 Differences between CIQ Entity Schemas used in Option 1 and**
1797 **Option 2**

1798 The key difference between the two types of CIQ entity schemas (Option 1 and Option 2) are the
1799 additional metadata information for information item values in XML instances for Option 2. This metadata
1800 information is defined as OPTIONAL attributes. It is not mandatory to have instance level metadata, but
1801 having it allows an instance to disambiguate a code value that might be the same value from two different
1802 lists. An application interpreting a given information item that has different values from different lists MAY
1803 need the user to specify some or the entire list metadata from which the value is found, especially if the
1804 value is ambiguous.

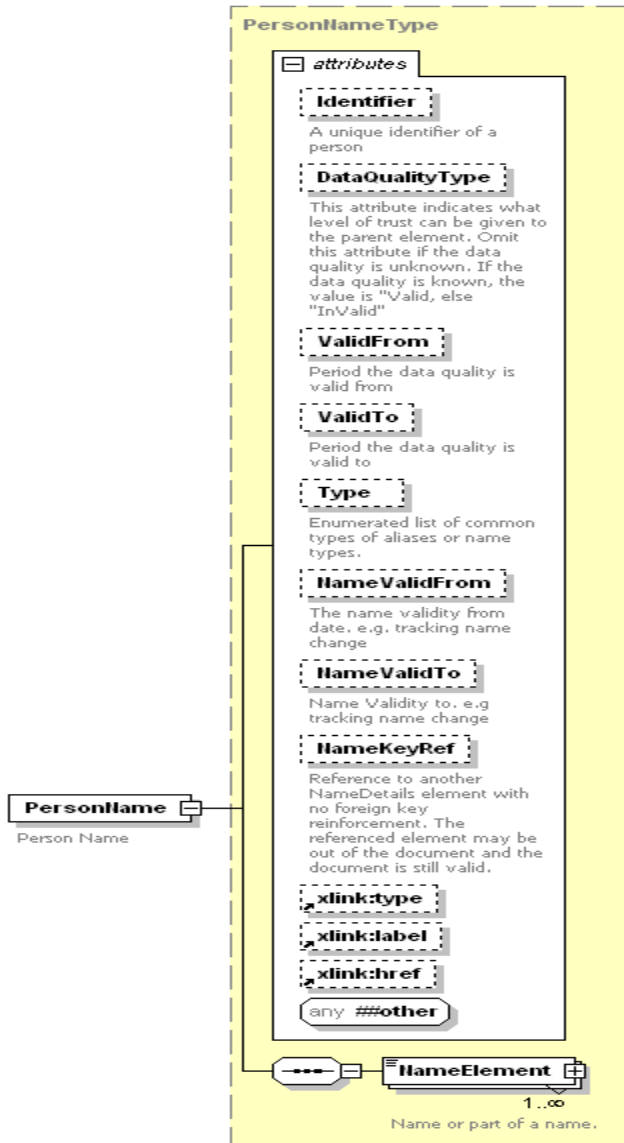
1805 Four types metadata attributes are used in Option 2 entity schema attributes that reference code lists and
1806 they are:

- 1807 • *Ref* – corresponds to genericcode <ShortName> reference
- 1808 • *Ver* – corresponds to genericcode <Version> version of the file
- 1809 • *URI* – corresponds to genericcode <CanonicalUri> abstract identifier for all versions of the code list
- 1810 • *VerURI* – corresponds to genericcode <CanonicalVersionUri> abstract identifier for this version of the
1811 code list

1812 For detailed explanation of metadata information, read CVA document ([http://www.oasis-](http://www.oasis-open.org/committees/document.php?document_id=21324)
1813 [open.org/committees/document.php?document_id=21324](http://www.oasis-open.org/committees/document.php?document_id=21324))

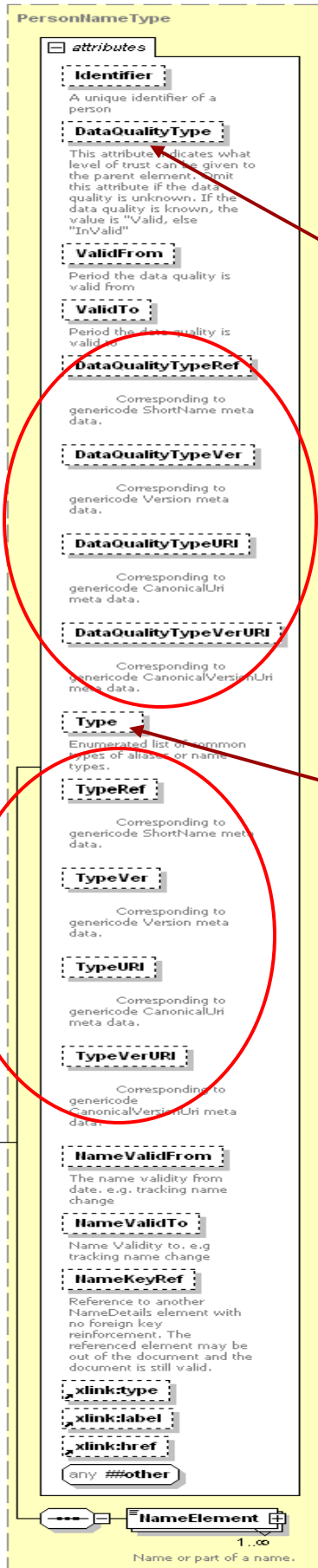
1814 The figure below shows “PersonName” element in Option 1 (using *xNL-types.xsd* for all Name entity
1815 associated code lists) of *xNL.xsd*:

1816



1817

1818 The figure below shows *PersonName* element in Option 2 (using genericcode for Name entity associated
 1819 code lists) of *xNL.xsd* with metadata information for genericcode based code lists:



PersonName
Person Name

Metadata Information for "DataQualityType" attribute that refers to genericcode "DataQualityEnumeration.gc" file

Metadata Information for "Type" attribute that refers to genericcode "PersonNameEnumeration.gc" file

1821 **7.4.1 Compatibility between XML documents produced using Option 1 and**
1822 **Option 2 CIQ XML Schemas**

1823 XML document instances that conform to CIQ XML schemas of Option 1 SHOULD validate against the
1824 CIQ XML schemas of Option 2 without any changes to the XML document. This MAY not true vice versa
1825 as Option 2 CIQ XML schemas provide “metadata attributes” to support genericcode and these attributes
1826 MAY be defined in the XML document instance. If these attributes are not defined in the XML document
1827 instance, then validation of the XML document instance against the CIQ XML Schemas of Option 1
1828 SHOULD be successful.

1829 **7.4.2 Which Code List Package to Use? Option 1 or Option 2?**

1830 User MUST use Option 1 or Option 2, but MUST NOT use both at the same time. The choice of the
1831 Option to use is entirely dependent on user specific requirements.

1832 8 Data Exchange and Interoperability

1833 OASIS CIQ TC defines data/information interoperability as follows:

1834 **“Get the *right data* to the *right place* at the *right time* in the *right format* with the *right quality* with**
1835 **the *right security* in the *right context* and with the *right governance* to applications, processes, or**
1836 **users”**

1837 It is the view of the CIQ committee that to enable interoperability of data/information between parties, the
1838 best solution is to parse the data elements into its atomic elements thereby preserving the semantics and
1839 quality of data. By this way the parties involved in data exchange will be in the best position to understand
1840 the semantics and quality of data thereby minimising interoperability issues. How the data will be
1841 exchanged between parties, whether in parsed or unparsed structure, must be negotiated between the
1842 parties to enable interoperability.

1843 One cannot expect interoperability to occur automatically without some sort of negotiation between
1844 parties (e.g. Information Exchange Agreement, whether internal or external to an organisation) involved in
1845 data exchange. Once information exchange agreements between parties are in place, then the
1846 data/information exchange process can be automated. Moreover, the entire information exchange and
1847 interoperability process SHOULD be managed through an effective governance process which SHOULD
1848 involve all the parties involved in the information exchange process. This enables effective and efficient
1849 management of any change to the information exchange process in the future.

1850 8.1 Data Interoperability Success Formula

1851 We at OASIS CIQ TC strongly believe in the following “Data Interoperability Success Formula”:

1852 **Data Interoperability = Open Data Architecture + Open Data Integration + Data Quality +**
1853 **Open Data Standards + Data Semantics + Data Security + Data Governance**

1854 All components on the right hand side of the above formula are important for successful data
1855 interoperability. The term “Open” used here indicates artifacts that are independent of any proprietary
1856 solution (e.g. open industry artifacts or artifacts that are open within an enterprise).

1857 8.2 Information Exchange Agreement – Guidelines

1858 To ensure interoperability of CIQ represented data/information between applications/business systems
1859 (whether internal to the organisation or external to the organisation) it is strongly advised that an
1860 information exchange agreement/specification for CIQ SHOULD be in place. This agreement/specification
1861 SHOULD outline in detail the customisation of CIQ specifications.

1862 Following are the features of CIQ specifications that assist in customisation of the specifications to meet
1863 specific application or data exchange requirements, and the details of customisation SHOULD be
1864 documented and agreed (if involving more than one party in data exchange) at application/system design
1865 time to enable automating interoperability of information/data represented using CIQ specifications at
1866 application/system run time:

- 1867 • List of all elements of CIQ XML Schemas that SHOULD be used in the exchange. This includes
1868 details of which elements are mandatory and which elements are OPTIONAL
- 1869 • List of all attributes of CIQ XML Schemas that SHOULD be used in the exchange. This includes
1870 details of which attributes are mandatory and which attributes are OPTIONAL
- 1871 • The approach that will be used for Code Lists (Option 1 or Option 2)
- 1872 • The code list values that SHOULD be used for each CIQ code lists. This includes updating the default
1873 XML Schemas for code lists (Option 1) with the values to be used and updating the default
1874 genericcode based code lists (Option 2) with the values to be used. These code list files SHOULD then
1875 be implemented by all applications/systems involved in data exchange. If genericcode based code list

1876 approach (Option 2) is used, then the XSLTs for value validation SHOULD be generated and
1877 implemented by all applications/systems involved in data exchange.

- 1878 • Whether xLink or Key Reference SHOULD be used to reference party, name or address, and the
1879 details
- 1880 • Whether XML schema SHOULD be extended by using new attributes from a non-target namespace
1881 and if so, details of the additional attributes
- 1882 • Whether business rules SHOULD be defined to constrain the CIQ XML schemas and if so, details of
1883 the business rules that SHOULD be implemented consistently by all applications/systems involved in
1884 data exchange

1885 Once the agreement is implemented, it is vital that the agreement SHOULD be governed through a
1886 governance process to manage change effectively and efficiently. All parties involved in the data
1887 exchange process SHOULD be key stakeholders of the governance process.

1888

1889 9 Conformance

1890 The keywords “MUST”, “MUST NOT”, “SHOULD”, “SHOULD NOT”, “MAY” and “OPTIONAL” interpreted
1891 as described in [RFC2119] are used as the conformance clauses throughout this document.

1892 9.1 Conformance Clauses

1893 9.1.1 Specifications Schema Conformance

1894 Implementation of CIQ Specifications namely the XML Schemas (*xNL.xsd*, *xAL.xsd*, *xNAL.xsd*, and
1895 *xPIL.xsd*) MUST conform to the specifications if the implementation conforms to as stated in section 3.12.

1896 9.1.2 Specifications Schema Extensibility Conformance

1897 Implementation of CIQ Specifications namely the XML Schemas (*xNL.xsd*, *xAL.xsd*, *xNAL.xsd*, and
1898 *xPIL.xsd*) by extending them MUST conform as stated in section 3.9.

1899 9.1.3 Specifications Code List Schema Customisation Conformance

1900 Customisation of the Code List XML Schemas (*xNL-types.xsd*, *xAL-types.xsd*, *xNAL-types.xsd*, and *xPIL-*
1901 *types.xsd*) using Option 1 MUST be well formed. Changes to the default values provided as part of the
1902 specifications is OPTIONAL and MAY be modified by the user.

1903 9.1.4 Interoperability Conformance

1904 Implementation of CIQ Specifications between two or more applications/systems or parties helps achieve
1905 interoperability if the implementation conforms to using the agreed conformance clauses as defined in
1906 sections 9.1.4.1, 9.1.4.2, 9.1.4.3, 9.1.4.4, 9.1.4.5, and 9.1.4.6.

1907 9.1.4.1 Interoperability Conformance – Using Elements and Attributes

1908 Implementation of elements and attributes of CIQ XML Schema enables interoperability if the following
1909 conditions are agreed by two or more parties involved in data exchange and are met:

- 1910 1. The OPTIONAL elements in the XML Schema that SHOULD be used for implementation and the
1911 OPTIONAL elements in the XML Schema that SHOULD be ignored. See section 8.2.
- 1912 2. The OPTIONAL attributes in the XML Schema that SHOULD be used for implementation and the
1913 OPTIONAL attributes in the XML Schema that SHOULD be ignored. See section 8.2 .

1914 9.1.4.2 Interoperability Conformance – Extending the Schema

1915 Implementation of the CIQ schema by extending it SHOULD be agreed and managed between two or
1916 more parties involved in the data exchange and MUST be conformed to in order to achieve
1917 interoperability as stated in section 3.9.

1918 9.1.4.3 Interoperability Conformance – Using Code Lists

1919 Implementation of a Code List approach SHOULD be agreed and conformance to the selected approach
1920 between two or more parties involved in the data exchange MUST be achieved in order to ensure
1921 interoperability and this is stated in section 3.4.

1922 9.1.4.4 Interoperability Conformance – Customising the Code Lists

1923 Implementation of the Code List values SHOULD be agreed between two or more parties involved in the
1924 data exchange and MUST be conformed to as agreed in order to ensure interoperability as stated in
1925 section 3.4.

1926 **9.1.4.5 Interoperability Conformance – Customising the Schema**

1927 Customisation of the schema SHOULD be achieved by the following ways:

1928 1. Using Code List values

1929 2. Defining new business rules to constraint the schema

1930 Implementation of the above approaches SHOULD be agreed between two or more parties involved in
1931 the data exchange and MUST be conformed to in order to achieve interoperability as stated in section
1932 3.13.

1933 **9.1.4.6 Interoperability Conformance – Data/Information Exchange Agreement**

1934 Implementation and conformance of the implementation to the agreed Data/Information Exchange
1935 Agreement between two or more parties involved in the data exchange MUST be achieved to ensure
1936 interoperability as stated in section 8.2.

1937

1938 **10 Miscellaneous**

1939 **10.1 Documentation**

1940 Although, all schema files are fully documented using XML Schema annotations it is not always
1941 convenient to browse the schema itself. This specification is accompanied by a set of HTML files auto
1942 generated by XML Spy. Note that not all information captured in the schema annotation tags is in the
1943 HTML documentation.

1944 **10.2 Examples**

1945 Several examples of instance XML documents for name, address and party schemas are provided as
1946 XML files. The examples are informative and demonstrate the application of this Technical Specification.

1947 The example files and their content are being constantly improved and updated on no particular schedule.

1948 **10.3 Contributions from Public**

1949 OASIS CIQ TC is open in the way it conducts its business. We welcome contributions from public in any
1950 form. Please, use "Send A Comment" feature on CIQ TC home page ([http://www.oasis-](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ciq)
1951 [open.org/committees/tc_home.php?wg_abbrev=ciq](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ciq)) to tell us about:

- 1952 • errors, omissions, misspellings in this specification, schemas or examples
- 1953 • your opinion in the form of criticisms, suggestions, comments, etc
- 1954 • willingness to contribute to the work of CIQ TC by becoming a member of the TC
- 1955 • willingness to contribute indirectly to the work of CIQ TC
- 1956 • provision of sample data that can be used to test the specifications
- 1957 • implementation experience
- 1958 • etc.

1959

1960 **11 Change Log**

1961 The major change to this specification from its earlier release in November 2007 is fix to xAL V3.0
1962 schema. Details about the issue and changes to the xAL schema are explained in the following document
1963 that is provided as part of this release package:

1964 Document Name: "CIQ Specification V3.0 – Address Schema (xAL.xsd) Errata", 19 March 2008

1965 File Name: ciq-xal-errata (file types: html, pdf or doc)

1966

A. Acknowledgements

1967 The following individuals have participated in the creation of version 3.0 of CIQ specifications and are
1968 gratefully acknowledged:

1969 **Participants:**

1970

Colin Wallis	New Zealand Government	Voting Member, CIQ TC
David Webber	Individual	Voting Member, CIQ TC
Fulton Wilcox	Colts Neck Solutions LLC	Voting Member, CIQ TC
Graham Lobsey	Individual	Voting Member, CIQ TC
Joe Lubenow	Individual	Voting Member, CIQ TC
John Glaubitz	Vertex, Inc	Voting Member, CIQ TC
Michael Roytman	Vertex, Inc	Voting Member, CIQ TC
Ram Kumar	Individual	Chair and Voting Member, CIQ TC
George Farkas	XBI Software, Inc	Former Member, CIQ TC
Hidajet Hasimbegovic	Individual	Former Member, CIQ TC
John Putman	Individual	Former Member, CIQ TC
Mark Meadows	Microsoft Corporation	Former Member, CIQ TC
Max Voskob	Individual	Former Member, CIQ TC
Robert James	Individual	Former Member, CIQ TC

1971

1972 OASIS CIQ Technical Committee (TC) sincerely thanks the public (this includes other standard groups,
1973 organisations and end users) for their continuous feedback and support that helps the TC to work toward
1974 improving the CIQ specifications.

1975 Special thanks to Mr.Ken Holman, Chair of OASIS Code List TC ([http://www.oasis-](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=codelist)
1976 [open.org/committees/tc_home.php?wg_abbrev=codelist](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=codelist)) for his support, guidance and genericode
1977 implementation assistance to the TC in releasing the OASIS Code List version of CIQ V3.0 XML
1978 Schemas.

1979 Special thanks to Mr.Hugh Wallis, Director of Standards Development of extensible Business Reporting
1980 Language (xBRL) International Standards Group (<http://www.xbrl.org>) for working closely with the CIQ TC
1981 in jointly implementing W3C xLink specification that is now used by both xBRL and CIQ Specifications to
1982 enable interoperability between the two specifications.

1983 Special thanks to Mr.Carl Reed, Chief Technology Officer of Open Geospatial Consortium (OGC –
1984 <http://www.opengeospatial.org>) for his guidance and assistance to the TC in referencing the work of OGC
1985 on GeoRSS and Geo-Coordinates for addresses/locations as part of CIQ Address Specifications.

1986 OASIS CIQ TC also acknowledges the contributions from other former members of the TC since its
1987 inception in 2000.

1988

1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006

B. Intellectual Property Rights, Patents, Licenses and Royalties

CIQ TC Specifications (includes documents, schemas and examples¹ 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.

¹xAL-AustralianAddresses.xml

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

²xAL-InternationalAddresses.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 and was developed jointly with OASIS CIQ TC.

2007

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
V3.0 PRD 02 R1	18 September 2007	Ram Kumar	Inclusion of comments from Public Review 02
V3.0 CS	15 November 2007	Ram Kumar	TC Approved Committee Specification
V3.0 CD 02	18 March 2008	Ram Kumar	Inclusion of the xAL Schema Errata
V3.0 PRD 03	08 April 2008	Ram Kumar	Public Review Draft for 15 days review
V3.0 CS 02	15 September 2008	Ram Kumar	TC Approved Committee Specification 02

2008