



CIQ TC Specifications

Customer Information Quality Technical Committee

Name, Address and Party Information Technical Specification

Version 3.0 (draft)

Date Created: 15 August 2004

Last Updated: 30 November 2005

Editors

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

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

Contributors

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

Abstract

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

Intellectual Property Rights, Patents, Licenses and Royalties

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

¹**xAL-Australia.XML**

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

²**xAL-international.xml**

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

TABLE OF CONTENTS

1	SCHEMA DESIGN APPROACH IN VERSION 3.0	5
1.1	VERSION 3.0 SCHEMA FILES	5
1.2	FORMAL DESIGN REQUIREMENTS FOR VERSION 3.0	5
1.3	MAJOR ENTITIES	6
1.4	COMMON APPROACHES	6
1.5	NAMESPACES	6
1.6	OTHER SPECIFICATIONS	6
2	ENTITY “NAME”	7
2.1	SEMANTICS OF “NAME”	7
2.2	DATA TYPES	9
2.3	ENUMERATIONS	9
2.4	ORDER OF ELEMENTS AND PRESENTATION	10
2.5	DATA MAPPING	10
2.6	DATA QUALITY	12
2.6.1	<i>Data quality verification and trust</i>	12
2.6.2	<i>Data validation</i>	12
2.7	EXTENSIBILITY	12
2.7.1	<i>Practical applications</i>	13
2.8	LINKING AND REFERENCING	13
2.9	ID ATTRIBUTE	14
2.10	SCHEMA CUSTOMIZATION GUIDELINES	14
2.10.1	<i>Namespace</i>	14
2.10.2	<i>Reducing the structure</i>	14
2.10.3	<i>Customizing the enumerations</i>	15
2.10.4	<i>Implications</i>	16
3	ENTITY “ADDRESS”	17
3.1	SEMANTICS OF “ADDRESS”	17
3.2	GEO-COORDINATES	18
3.3	DATA TYPES	19
3.4	ENUMERATIONS	19
3.5	ORDER OF ELEMENTS AND PRESENTATION	19
3.6	DATA MAPPING	19
3.7	DATA QUALITY	20
3.8	EXTENSIBILITY	20
3.9	LINKING AND REFERENCING	20
3.10	SCHEMA CUSTOMIZATION	20
4	COMBINATION OF “NAME” AND “ADDRESS”	21
4.1	USE OF ELEMENT XNAL:RECORD	21
4.2	USE OF ELEMENT XNAL:POSTAL LABEL	21
5	ENTITY “PARTY”	23
5.1	DEALING WITH JOINT PARTY NAMES	25
5.2	DATA TYPES	26
5.3	ENUMERATIONS	26
5.4	ORDER OF ELEMENTS AND PRESENTATION	26
5.5	DATA MAPPING	26
5.6	DATA QUALITY	26
5.7	EXTENSIBILITY	26
5.8	LINKING AND REFERENCING	26
5.9	SCHEMA CUSTOMIZATION	26

6 MISCELLANEOUS	27
6.1 DOCUMENTATION.....	27
6.2 EXAMPLES.....	27
6.3 CONTRIBUTIONS FROM PUBLIC.....	27
APPENDIX A. NOTICES.....	28

1 Schema design approach in version 3.0

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

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

7 1.1 Version 3.0 schema files

8 Following are the different schemas produced for version 3.0:

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

11 1.2 Formal design requirements for version 3.0

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

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

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

25 1.3 Major entities

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

- 29 • Name (see xNL.xsd, xNL-types.xsd)
- 30 • Address (see xAL.xsd, xAL-types.xsd)
- 31 • Name and Address combined (see xNAL.xsd)
- 32 • Personal details and specifics (see xPIL.xsd, xPIL-types.xsd)
- 33 • Organisation details and specifics (see xPIL.xsd, xPIL-types.xsd)
- 34 • Party Relationships (see xPRL.xsd and xLink.xsd)

35 1.4 Common approaches

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

41 1.5 Namespaces

42

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

43 1.6 Other specifications

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

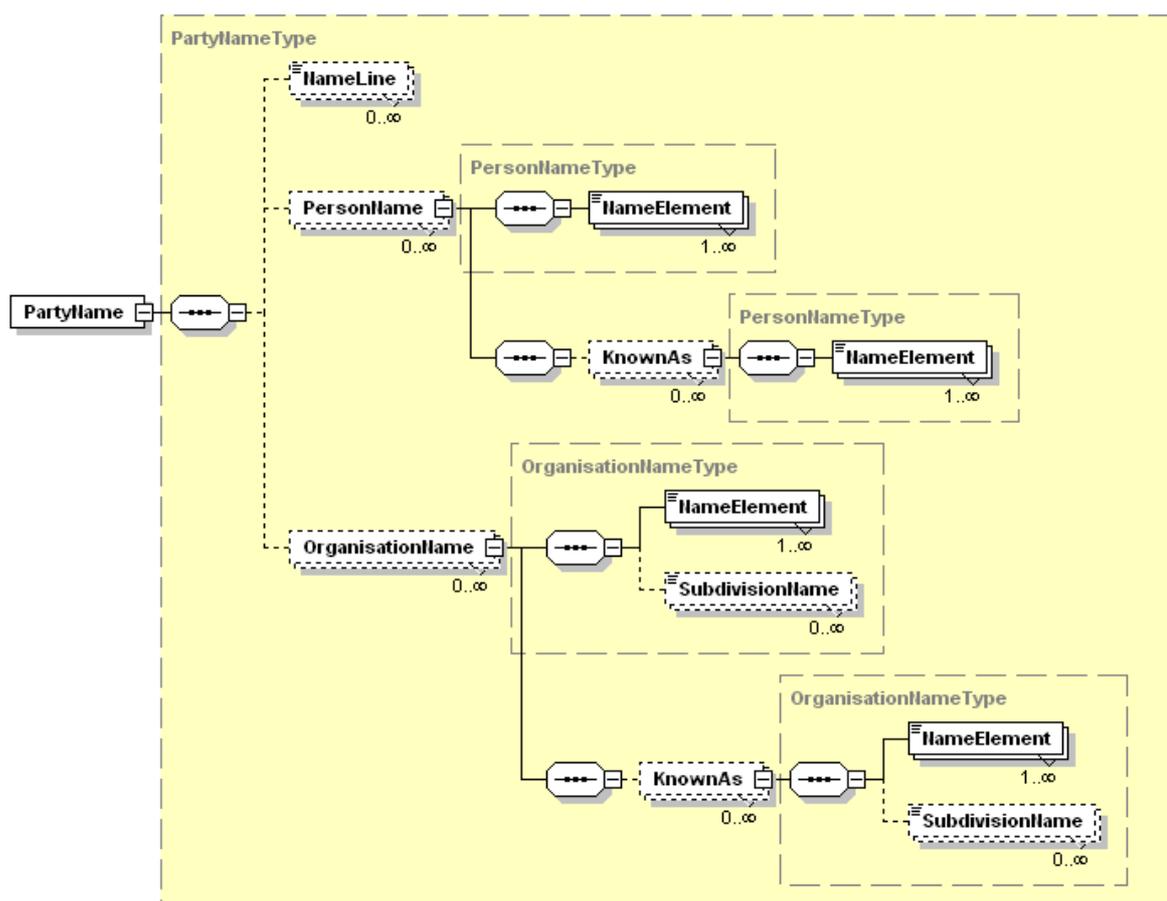
46 2 Entity “Name”

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

49 2.1 Semantics of “Name”

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

55



56

57

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

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

63

64

65

66 **Example 1 – no semantics**

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

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

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

77

78 **Example 2 – minimal semantics**

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

81 Person name:

```
82 <n:PartyName>
83   <n:PersonName>
84     <n:NameElement>Mr Jeremy Apatuta Johnson</n:NameElement>
85   </n:PersonName>
86 </n:PartyName>
```

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

89

90 Organisation name:

```
91 <n:PartyName>
92   <n:OrganisationName>
93     <n:NameElement>Khandallah Laundering Ltd.</n:NameElement>
94   </n:OrganisationName>
95 </n:PartyName>
```

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

97

98 **Example 3 – full semantics**

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

```
102 <n:PartyName>
103   <n:PersonName>
104     <n:NameElement Abbreviation="true" ElementType="Title">Mr</n:NameElement>
105     <n:NameElement ElementType="FirstName">Jeremy</n:NameElement>
106     <n:NameElement ElementType="MiddleName">Apatuta</n:NameElement>
107     <n:NameElement ElementType="LastName">Johnson</n:NameElement>
108     <n:NameElement ElementType="GenerationIdentifier">III</n:NameElement>
109     <n:NameElement ElementType="GenerationIdentifier">Junior</n:NameElement>
110     <n:NameElement ElementType="Title">PhD</n:NameElement>
111   </n:PersonName>
112 </n:PartyName>
```

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

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

117

```

118 <xs:simpleType name="PersonNameElementsEnumeration">
119   <xs:restriction base="xs:string">
120     <xs:enumeration value="PrecedingTitle"/>
121     <xs:enumeration value="Title"/>
122     <xs:enumeration value="FirstName"/>
123     <xs:enumeration value="MiddleName"/>
124     <xs:enumeration value="LastName"/>
125     <xs:enumeration value="OtherName"/>
126     <xs:enumeration value="Alias"/>
127     <xs:enumeration value="GenerationIdentifier"/>
128   </xs:restriction>
129 </xs:simpleType>

```

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

135 2.2 Data types

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

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

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

141 2.3 Enumerations

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

151 Example – point-to-point

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

```

156 <xs:simpleType name="PersonNameElementsEnumeration">
157   <xs:restriction base="xs:string">
158     <xs:enumeration value="Title"/>
159     <xs:enumeration value="FirstName"/>
160     <xs:enumeration value="MiddleName"/>
161     <xs:enumeration value="LastName"/>
162   </xs:restriction>
163 </xs:simpleType>

```

164 Example – locale specific

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

```

166 <xs:simpleType name="PersonNameElementsEnumeration">
167   <xs:restriction base="xs:string">
168     <xs:enumeration value="Title"/>
169     <xs:enumeration value="Name"/>
170     <xs:enumeration value="FathersName"/>
171     <xs:enumeration value="FamilyName"/>
172   </xs:restriction>
173 </xs:simpleType>

```

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

176 2.4 Order of elements and presentation

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

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

181 Example – normal order

182
 183 `Mr Jeremy Apatuta Johnson PhD`

184 could be presented as follows

```

185 <n:PartyName>
186   <n:PersonName>
187     <n:NameElement>Mr</n:NameElement>
188     <n:NameElement>Jeremy</n:NameElement>
189     <n:NameElement>Apatuta</n:NameElement>
190     <n:NameElement>Johnson</n:NameElement>
191     <n:NameElement>PhD</n:NameElement>
192   </n:PersonName>
193 </n:PartyName>
    
```

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

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

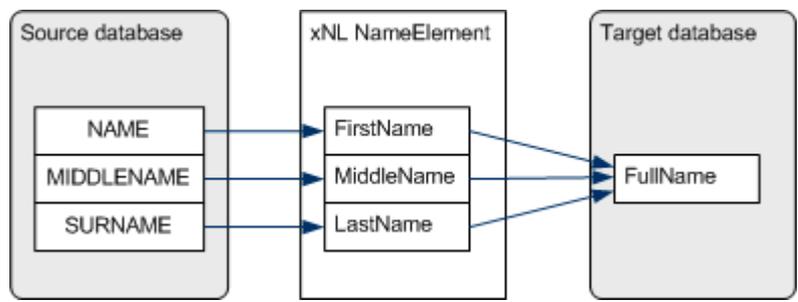
197 2.5 Data mapping

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

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

207 Example – complex-to-simple mapping

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



211
 212
 213

Name, Address and Party Information

214 **Source database**

NAME	MIDDLENAME	SURNAME
John	Anthony	Jackson

215

216 **xNL**

```

217 <n:PersonName>
218 <n:NameElement n:ElementType="FirstName">John</n:NameElement>
219 <n:NameElement n:ElementType="MiddleName">Anthony</n:NameElement>
220 <n:NameElement n:ElementType="LastName">Jackson</n:NameElement>
221 </n:PersonName>
    
```

222

223 **Target database**

FULLNAME
John Anthony Jackson

224

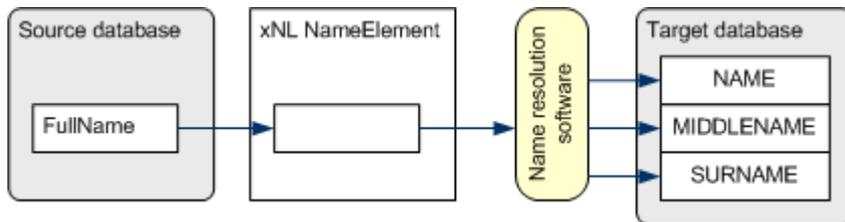
225 This type of mapping does not present a major challenge as it is a direct mapping from source
 226 to xNL and then concatenating the data values to form the full name to be stored in a
 227 database field/column.

228

229 **Example – simple-to-complex mapping**

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

232



233

234

235 **Source database**

FULLNAME
John Anthony Jackson

236

237 **xNL**

```

238 <n:PersonName>
239 <n:NameElement>John Anthony Jackson</n:NameElement>
240 </n:PersonName>
    
```

241

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

244

245

246

247

248 Target database

249

NAME	MIDDLENAME	SURNAME
John	Anthony	Jackson

250

251 2.6 Data quality

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

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

258 Example – data quality

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

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

264

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

268 2.6.1 Data quality verification and trust

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

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

272 2.6.2 Data validation

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

275 2.7 Extensibility

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

278 All elements share the same declaration:

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

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

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

284

285

286 **2.7.1 Practical applications**287 **System-specific identifiers**

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

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

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

```
300
301 <n:PartyName xmlns:b="urn:acme.org:corporate:IDs" b:PartyID="123445" />
```

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

303 **Additional metadata**

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

```
306 <n:PartyName xmlns:x="urn:acme.org:corporate" x:OperatorID="buba7">
307   .....
```

```
308
309 <n:PartyName xmlns:b="urn:acme.org:corporate" >
310   <n:PersonName>
311     <n:NameElement b:Corrected="true">John Johnson</n:NameElement>
312   </n:PersonName>
313 </n:PartyName>
```

314

315 **2.8 Linking and referencing**

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

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

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

323

```
324 <a:Contacts
325   xmlns:a="urn:acme.org:corporate:contacts"
326   xmlns:n="urn:oasis:names:tc:ciq:xsd:schema:xNL:3.0/20050427"
327   xmlns:xlink="http://www.w3.org/1999/xlink">
328   <n:PartyName xlink:href="http://example.org/party?id=123445" xlink:type="locator"/>
329   <n:PartyName xlink:href="http://example.org/party?id=83453485" xlink:type="locator"/>
330 </a:Contacts>
```

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

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

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

338 2.9 ID attribute

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

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

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

```
351 <n:PartyName n:ID="52F89CC0-5C10-4423-B367-2E8C14453926">
352   <n:PersonName>
353     <n:NameElement>Max Voskob</n:NameElement>
354   </n:PersonName>
355   <n:OrganisationName>
356     <n:NameElement>Khandallah Laundering Ltd.</n:NameElement>
357   </n:OrganisationName>
358 </n:PartyName>
```

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

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

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

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

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

368 2.10 Schema customization guidelines

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

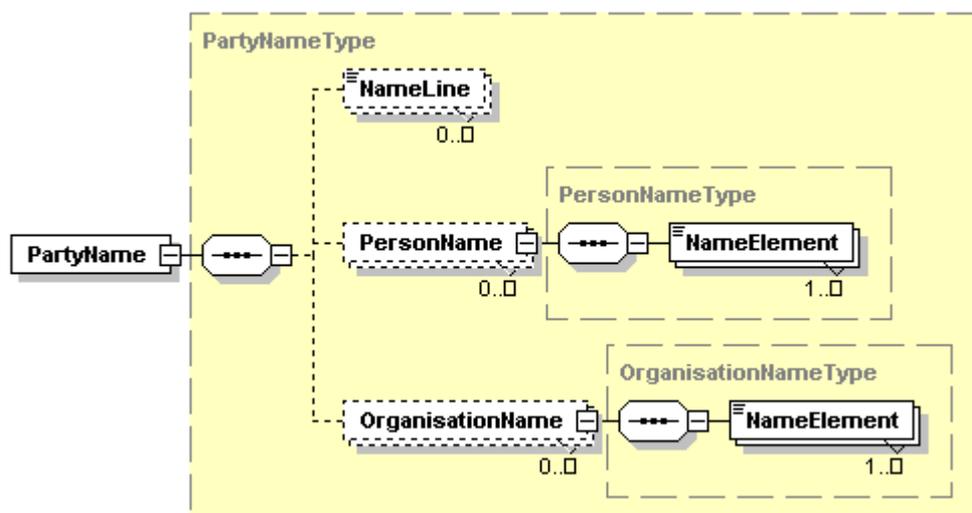
373 2.10.1 Namespace

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

376 2.10.2 Reducing the structure

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

378



379

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

- 382
- additional names (*KnownAs*), e.g. maiden name as part of *PartyName*
 - organisation subdivision hierarchy (*SubdivisionName*), e.g. faculty / school / department
- 383
384

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

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

389 2.10.3 Customizing the enumerations

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

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

394 Proprietary enumeration example

395

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

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

397
398
399
400
401
402
403
404
405

```

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

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

410 **2.10.4 Implications**

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

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

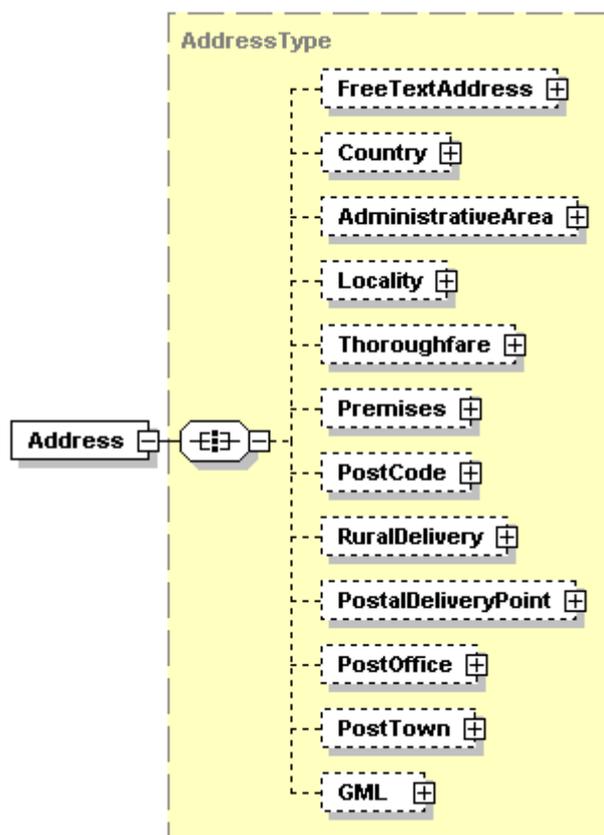
415 3 Entity “Address”

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

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

420 3.1 Semantics of “Address”

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



422

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

424 Example – free text

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

429

```
430 <a:Address>
431   <a:FreeTextAddress>
432     <a:AddressLine>Substation C</a:AddressLine>
433     <a:AddressLine >17 James Street</a:AddressLine >
434     <a:AddressLine>SPRINGVALE VIC 3171</a:AddressLine>
435   </a:FreeTextAddress>
436 </a:Address>
```

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

439

440 **Example – semi structured address**

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

```

443 <a:Address>
444   <a:AdministrativeArea>
445     <a:Name>WA</a:Name>
446   </a:AdministrativeArea>
447   <a:Locality>
448     <a:Name>OCEAN REEF</a:Name>
449   </a:Locality>
450   <a:Thoroughfare>
451     <a:NameElement>16 Patterson Street</a:NameElement>
452   </a:Thoroughfare>
453 </a:Address>

```

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

459

460 **Example – fully structured address**

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

```

463 <a:Address>
464   <a:AdministrativeArea>
465     <a:Name a:Abbreviation="true" a:NameType="state">VIC</a:Name>
466   </a:AdministrativeArea>
467   <a:Locality>
468     <a:Name>CLAYTON</a:Name>
469     <a:SubLocality>Technology Park</a:SubLocality>
470   </a:Locality>
471   <a:Thoroughfare>
472     <a:NameElement>Dandenong Road</a:NameElement>
473     <a:Number a:EnumeratedType="RangeFrom">200</a:Number>
474     <a:Number a:EnumeratedType="Separator">-</a:Number>
475     <a:Number a:EnumeratedType="RangeTo">350</a:Number>
476     <a:SubThoroughfare>
477       <a:NameElement>Fifth Avenue</a:NameElement>
478     </a:SubThoroughfare>
479   </a:Thoroughfare>
480   <a:Premise>
481     <a:NameElement>Toshiba Building</a:NameElement>
482   </a:Premise>
483   <a:PostalCode>
484     <a:Number>3168</a:Number>
485   </a:PostalCode>
486 </a:Address>

```

487 **3.2 Geo-coordinates**

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

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

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

- 495
- Be from the GML namespace

- 496 • Refer to finest level of address details available in the address structure *a:GML* belongs
497 to
- 498 • Be used unambiguously so that there is no confusion whether the coordinates belong
499 to the postal delivery point (e.g. Post Box) or a physical address (e.g. flat) as it is
500 possible to have both in the same address structure.
- 501 There is no restriction on the shape of the area *a:GML* can describe be it a point, polygon or
502 some other object.

503 3.3 Data types

- 504 All elements and attributes in *xAL* schema have strong data types.
- 505 All free-text values of elements (text nodes) and attributes are constrained by simple type
506 “*string*” defined in *xAL-types.xsd*. This type has a limit on the number of characters it may
507 contain.
- 508 Other XML Schema defined data types are also used throughout *xAL* namespace.

509 3.4 Enumerations

- 510 Use of enumerations is identical to use of enumerations for entity “*Name*”. Refer to section
511 2.3 Enumerations for more information.
- 512 Enumerations used in *xAL* reside in an “include” file *xAL-types.xsd*.

513 3.5 Order of elements and presentation

- 514 Order of address elements should be preserved for correct presentation in a fashion similar to
515 what is described in section 2.4 Order of elements and presentation.
- 516 Child elements of *a:Address* can appear in any order as members of *xs:all* grouping as in the
517 example below:

518 Example – order of second level elements in *xAL*

519
520
521
522

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

523 could be preserved and presented in XML as:

524
525
526
527
528
529
530

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

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

532 3.6 Data mapping

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

535 Example – normal order

536
537
538
539

23 Archer Street
Chatswood, NSW 2067
Australia

540 could be presented as follows

```
541 <a:Address>  
542   <a:FreeTextAddress>  
543     <a:AddressLine>23 Archer Street</a:AddressLine>  
544     <a:AddressLine>Chatswood, NSW 2067</a:AddressLine>  
545     <a:AddressLine>Australia</a:AddressLine>  
546   </a:FreeTextAddress>  
547 </a:Address>
```

548 and restored back to

```
549 23 Archer Street  
550 Chatswood, NSW 2067  
551 Australia
```

552 during data formatting exercise.

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

555 **3.7 Data quality**

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

558 **3.8 Extensibility**

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

560 **3.9 Linking and referencing**

561 All linking and referencing rules described in section 2.8 Linking and apply to entity
562 “*Address*”.

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

564 **3.10 Schema customization**

565 Schema customisation rules and concepts described in section 2.10 Schema customization
566 are fully applicable to entity “*Address*”.

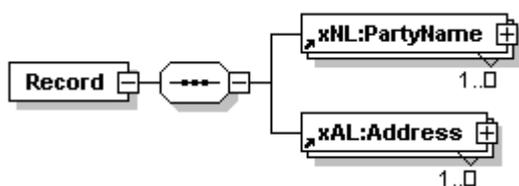
567 4 Combination of “Name” and “Address”

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

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

573 4.1 Use of element *xnal:Record*

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



576

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

581 Example

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

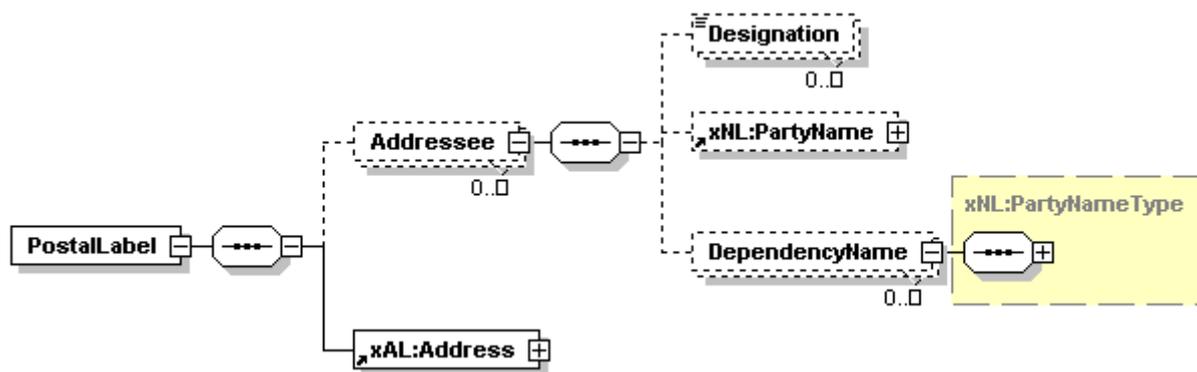
```

583 <xnal:Record>
584   <n:PartyName>
585     <n:NameLine>Mr H G Guy</n:NameLine>
586   </n:PartyName>
587   <a:Address>
588     <a:Locality>
589       <a:Name>Christchurch</a:Name>
590       <a:SubLocality>Redwood</a:SubLocality>
591     </a:Locality>
592     <a:Thoroughfare>
593       <a:Number>9</a:Number>
594       <a:NameElement>Uxbridge Street</a:NameElement>
595     </a:Thoroughfare>
596     <a:PostCode>
597       <a:Identifier>8005</a:Identifier>
598     </a:PostCode>
599   </a:Address>
600 </xnal:Record>
  
```

601 4.2 Use of element *xnal:PostalLabel*

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

Name, Address and Party Information



604

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

607

608 **Example**

609

```

610 Attention: Mr S Mart
611 Name Plate Engravers
612 The Emporium
613 855 Atawhai Drive
614 Atawhai
615 Nelson 7001
  
```

616

Translates into the following xNAL fragment:

617

```

618 <xnal:PostalLabel>
619   <xnal:Addressee>
620     <xnal:Designation>Attention: Mr S Mart</xnal:Designation>
621     <n:PartyName>
622       <n:NameLine>Name Plate Engravers</n:NameLine>
623     </n:PartyName>
624   </xnal:Addressee>
625   <a:Address>
626     <a:Locality>
627       <a:Name>Nelson</a:Name>
628       <a:SubLocality>Atawhai</a:SubLocality>
629     </a:Locality>
630     <a:Thoroughfare>
631       <a:NameElement>Atawhai Drive</a:NameElement>
632       <a:Number>855</a:Number>
633     </a:Thoroughfare>
634     <a:PostCode>
635       <a:Identifier>7001</a:Identifier>
636     </a:PostCode>
637   </a:Address>
</xnal:PostalLabel>
  
```

638

639

640

5 Entity “Party”

641

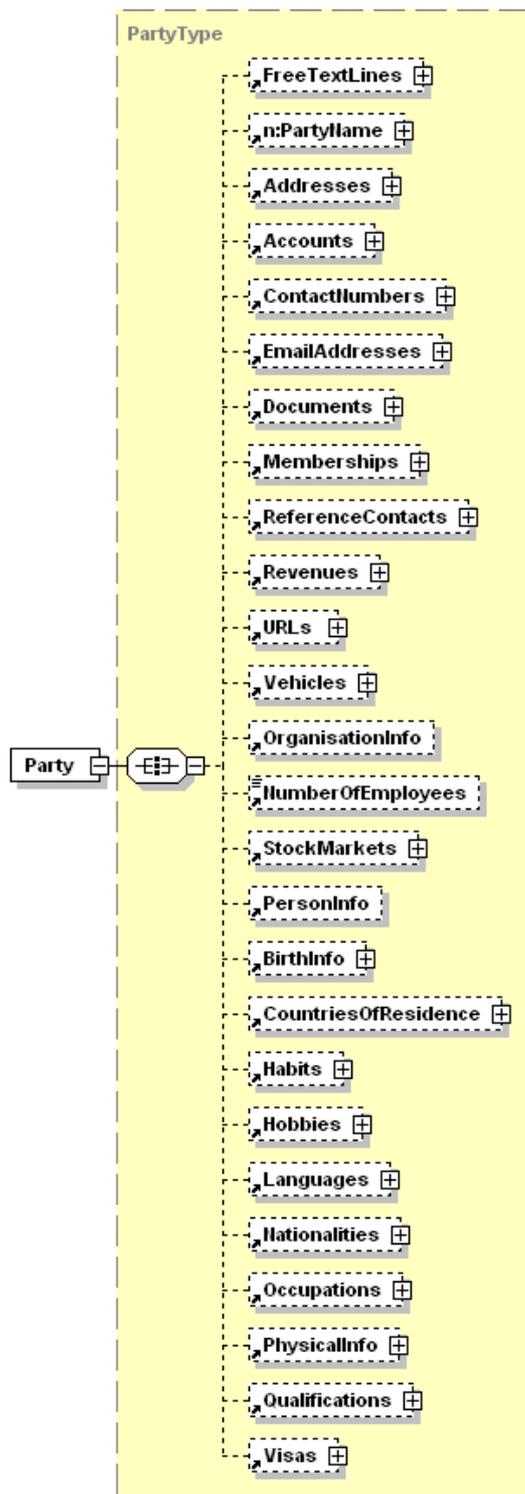
Entity “Party” encapsulates some most commonly used unique characteristics of *Person* or *Organisation*, such as name, address, personal details, contact details, body features, etc.

643

The diagram below shows the high level structure of *Party*. The full schema can be found in

644

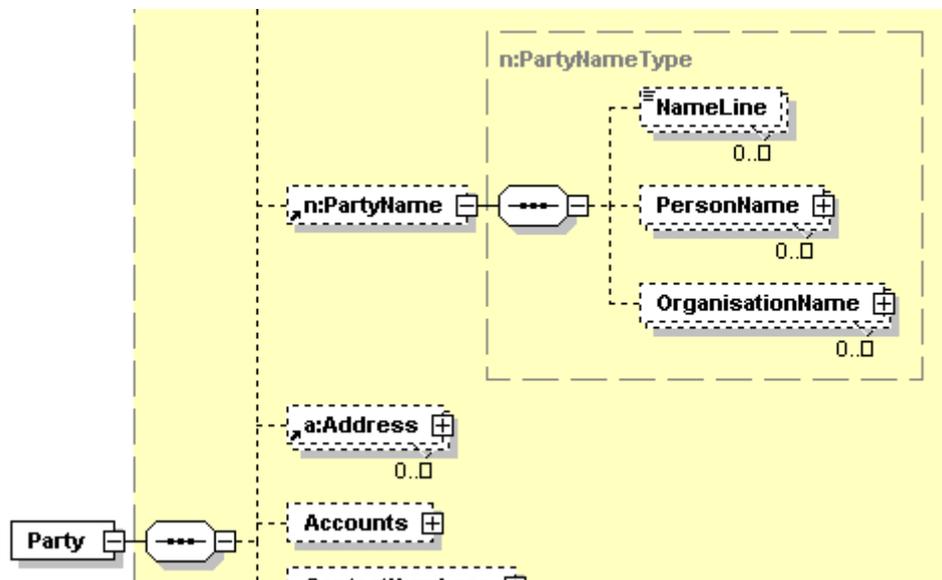
xPIL,xsd file with enumerations in *xPIL-types.xsd* file. See the sample XML files for examples.



645

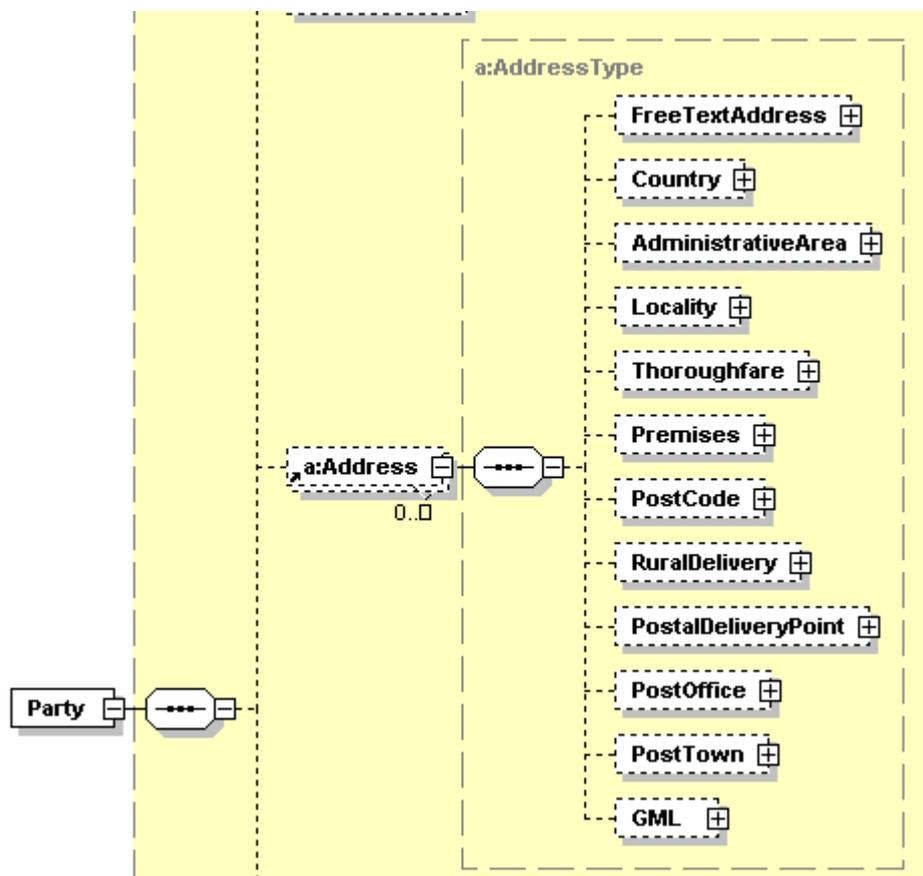
Name, Address and Party Information

- 646 The schema consists of top level containers that may appear in any order or be omitted. The
- 647 containers are declared globally and can be reused by other schemas.
- 648 The shared elements apply to *organisation* as well as *person*.
- 649 Name of the party reuses *PartyNameType* construct from xNL namespace as illustrated in the
- 650 following diagram:



- 651
- 652
- 653
- 654

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



655

Name, Address and Party Information

656

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

659 The following example illustrates use of some of party constructs

660

661 Example – qualification details

```
662 <p:Qualifications>
663   <p:Qualification>
664     <p:QualificationElement
665       p:ElementType="QualificationName">BComp.Sc.</p:QualificationElement>
666     <p:QualificationElement
667       p:ElementType="MajorSubject">Mathematics</p:QualificationElement>
668     <p:QualificationElement
669       p:ElementType="MinorSubject">Statistics</p:QualificationElement>
670     <p:QualificationElement p:ElementType="Award">Honours</p:QualificationElement>
671     <p:InstitutionName>
672       <n:NameLine>University of Technology Sydney</n:NameLine>
673     </p:InstitutionName>
674   </p:Qualification>
675 </p:Qualifications>
```

676

677 Example – birth details

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

679

680 Example – driver license

```
681 <p:Document p:ValidTo="2004-04-22T00:00:00">
682   <p:IssuePlace>
683     <a:Country>
684       <a:Name>Australia</a:Name>
685     </a:Country>
686     <a:AdministrativeArea>
687       <a:Name>NSW</a:Name>
688     </a:AdministrativeArea>
689   </p:IssuePlace>
690   <p:DocumentElement p:ElementType="DocumentID">74183768C</p:DocumentElement>
691   <p:DocumentElement p:ElementType="DocumentType">Driver License</p:DocumentElement>
692   <p:DocumentElement p:ElementType="Priveledge">Silver</p:DocumentElement>
693   <p:DocumentElement p:ElementType="Restriction">Car</p:DocumentElement>
694 </p:Document>
```

695

696 Example – contact phone number

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

703

704 5.1 Dealing with Joint Party Names

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

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

713 5.2 Data types

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

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

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

719 5.3 Enumerations

720 Use of enumerations is identical to use of enumerations for entity “Name”. Refer to section
721 2.3 Enumerations for more information.

722 Enumerations used in *xPIL* reside in an “include” *xPIL-types.xsd*.

723 5.4 Order of elements and presentation

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

727 5.5 Data mapping

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

730 5.6 Data quality

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

733 5.7 Extensibility

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

735 5.8 Linking and referencing

736 All linking and referencing rules described in section 2.8 Linking and apply to entity “Party”.

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

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

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

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

747 5.9 Schema customization

748 Schema customisation rules and concepts described in section 2.10 Schema customization
749 are fully applicable to entity “Party”.

750 **6 Miscellaneous**

751 **6.1 Documentation**

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

756 **6.2 Examples**

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

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

762 **6.3 Contributions from Public**

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

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

773

774 **Appendix A. Notices**

775 *Copyright © OASIS Open 2005. All Rights Reserved.*

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

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

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

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

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

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

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

818