IMPORTANT NOTE: This document contains or relates to ACORD Standard(s) for Cross Domain. You are not authorized to use the ACORD Standard contained in this document unless you have accepted the terms and conditions of the Standards License accessible at http://legal.acord.org/standards_license.htm. To gain such authorization, please go to that site and, if you agree with the terms and conditions of the Standards License, enter whatever information is called for, if any, and click on 'Accept'.
# TABLE OF CONTENTS

1. **INTRODUCTION** ......................................................................................................................... 8  
   1.1 Purpose of this Guide ..................................................................................................... 8  
   1.2 Audience .................................................................................................................. 8  
   1.3 Scope ..................................................................................................................... 8  
   1.4 Compatibility With UN/CEFACT NDR ......................................................................... 9  
   1.5 Notational Conventions ................................................................................................. 9  
   1.6 Related Documents ...................................................................................................... 10  
   1.7 Conformance ............................................................................................................... 11  
   1.8 Governing Principles ..................................................................................................... 11  

2. **RELATIONSHIP TO THE FRAMEWORK AND UNDERLYING STANDARDS** ...................... 12  
   2.1 Relationship to SOA and Web Services..................................................................... 12  
   2.1.1 Relationship to SOA ............................................................................................. 12  
   2.1.2 Relationship to Web Services .................................................................................. 13  
   2.2 Relationship to UML ...................................................................................................... 13  
   2.2.1 Assumed Modeling Principles................................................................................. 13  
   2.2.1.1 Definitions......................................................................................................... 14  
   2.2.1.2 Main Assumptions............................................................................................. 14  
   2.2.1.3 More Details....................................................................................................... 14  
   2.2.2 UML Message Class Template ............................................................................... 15  
   2.3 Relationship to the Core Component Technical Specification............................ 16  

3. **GENERAL XML ARCHITECTURE** .............................................................................................. 18  
   3.1 Versioning and Backwards Compatability ............................................................... 18  
   3.1.1 Versioning Scheme ................................................................................................... 18  
   3.1.2 Major Version ............................................................................................................. 18  
   3.1.3 Minor Version ............................................................................................................. 19  
   3.1.3.1 Adding Redundant Elements or Constructs ................................................... 19  
   3.1.3.2 Deprecation and Deletion ............................................................................... 20  
   3.1.4 Revision ....................................................................................................................... 20  
   3.2 Namespace Architecture ........................................................................................... 20  
   3.2.1 Namespace Segmentation .................................................................................... 21  
   3.2.1.1 Process Domains............................................................................................... 22
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1.2</td>
<td>Data Domain</td>
<td>22</td>
</tr>
<tr>
<td>3.2.1.3</td>
<td>Documentation Domain</td>
<td>22</td>
</tr>
<tr>
<td>3.2.2</td>
<td>ACORD Namespace Scheme</td>
<td>23</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Declaring Namespace</td>
<td>25</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Namespace Persistence</td>
<td>25</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Namespace Uniform Resource Identifiers</td>
<td>25</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Namespace Constraint</td>
<td>26</td>
</tr>
<tr>
<td>3.2.7</td>
<td>List of ACORD Standard Namespaces</td>
<td>26</td>
</tr>
<tr>
<td>3.3</td>
<td>Schema Management and Versioning</td>
<td>27</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Schema Set</td>
<td>27</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Schema Modules Per Namespace</td>
<td>28</td>
</tr>
<tr>
<td>3.3.2.1</td>
<td>Process Domains - Message Schema Sets</td>
<td>28</td>
</tr>
<tr>
<td>3.3.2.2</td>
<td>Data Domain - Reusable Data Components Set</td>
<td>28</td>
</tr>
<tr>
<td>3.3.2.3</td>
<td>Documentation Domain - Core Data Types and Annotations</td>
<td>29</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Schema and Component Versioning</td>
<td>29</td>
</tr>
<tr>
<td>3.3.3.1</td>
<td>Schema Versioning</td>
<td>29</td>
</tr>
<tr>
<td>3.3.3.2</td>
<td>Schema File Names</td>
<td>30</td>
</tr>
<tr>
<td>3.3.3.2.1</td>
<td>Schema Version Indication</td>
<td>31</td>
</tr>
<tr>
<td>3.3.3.3</td>
<td>Component Versioning</td>
<td>32</td>
</tr>
<tr>
<td>3.3.3.3.1</td>
<td>Use of Schema Annotations for Component Versioning</td>
<td>32</td>
</tr>
<tr>
<td>3.3.3.4</td>
<td>Indicating Schema Version And Message Component Version In Message Instance</td>
<td>33</td>
</tr>
<tr>
<td>3.3.3.5</td>
<td>Schema Slicing/Slimming</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>Schema Design And Generation</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>Schema Design Style</td>
<td>34</td>
</tr>
<tr>
<td>4.1.1</td>
<td>General Principles</td>
<td>34</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Reusability of Schema Objects</td>
<td>34</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Hybrid Design Pattern</td>
<td>35</td>
</tr>
<tr>
<td>4.1.4</td>
<td>XML Referencing</td>
<td>36</td>
</tr>
<tr>
<td>4.1.4.1</td>
<td>Defining And Referencing Objects in a Document Instance</td>
<td>36</td>
</tr>
<tr>
<td>4.1.4.1.1</td>
<td>Structural Referencing</td>
<td>36</td>
</tr>
<tr>
<td>4.1.4.1.2</td>
<td>Dynamic Referencing</td>
<td>36</td>
</tr>
<tr>
<td>4.1.4.2</td>
<td>Schema Validation</td>
<td>36</td>
</tr>
<tr>
<td>4.1.4.2.1</td>
<td>Validation Requirements</td>
<td>36</td>
</tr>
<tr>
<td>4.1.4.2.2</td>
<td>Schema Identity Constraints</td>
<td>37</td>
</tr>
<tr>
<td>4.1.4.2.3</td>
<td>Usage of Key/KeyRef for Structural Referencing</td>
<td>37</td>
</tr>
</tbody>
</table>
C. SCHEMA MODULE TEMPLATES .............................................................................................................. 116

D. ANNOTATION TEMPLATES .................................................................................................................. 117
   Schema Element ........................................................................................................................................ 117
   Simple Type .............................................................................................................................................. 117
   Global Element ........................................................................................................................................ 118
   Complex Type and Contained Local Elements ....................................................................................... 118
   Complex Type and Contained Global Elements ..................................................................................... 119
   Complex Type and Referenced Global Elements .................................................................................... 120
   Code Lists and Declared Enumerations .................................................................................................. 121

E. VERSIONING POLICY .......................................................................................................................... 123
   Background ................................................................................................................................................ 123
      What is Compatibility .......................................................................................................................... 123
      Backward Compatibility ...................................................................................................................... 123
      Forward Compatibility ........................................................................................................................ 124
   Compatibility Policy ................................................................................................................................ 124
   Policy Implications .................................................................................................................................. 125
      Responsibilities of Implementers ........................................................................................................... 125
      Membership Responsibilities for Evolution .......................................................................................... 125
      Categorization of Maintenance Requests ............................................................................................ 126

F. REQUIREMENTS FOR STANDARD METHOD FOR DESIGNING, PUBLISHING AND MANAGING CUSTOMIZED SPECIFICATIONS ............................................................................................................. 131
   Requirements ........................................................................................................................................... 131
   Business Issues ....................................................................................................................................... 132
   Technical Issues ....................................................................................................................................... 132
1. INTRODUCTION

1.1 Purpose of this Guide

The ACORD Naming and Design Rules Specification documents the common XML architectural functionality, naming conventions, design rules, schema implementation and data types that are used in ACORD XML Specifications. This specification provides the information and rules necessary to define XML structures that will be reused across all domains at ACORD.

1.2 Audience

The primary audience for this document is the technical implementer, business analysts, and members of working groups who are responsible for designing ACORD XML Specifications. This specification provides the building blocks for developing other specifications at ACORD; as such it must be adopted and used for some business domain; by itself, this specification has no function.

1.3 Scope

This technical specification will form the basis for standards development of XML schemas based on information models developed in accordance with the ACORD Standard Framework.

Important factors have emerged from work done on the ACORD Standard Framework that impact the scope of the ACORD NDR work:

- As more work has been done on standardization of “business process services” it has become clearer that it would be impractical to create ACORD messages for every process that is defined and that instead implementers following a Service Oriented Architecture may need to generate their own messages based on ACORD XML data standards to exchange data between processes;

- In the course of formalizing its UML based data modeling methodology, ACORD has recognized the “Core Component” modeling developed by UN/CEFACT as a compatible basis for the development of the ACORD data dictionary;

- UN/CEFACT, OASIS/UBL and other standard development organizations have made significant progress in working together to come up with a single NDR approach that sets rules for generation of XML data standards directly from Core Component models.

These factors have pushed ACORD to extend the initial scope of the ACORD NDR project to address the issue of generating XML constructs from the ACORD data dictionary in a way that is compatible with XML generation from “Core Components” dictionaries.

This is an important opportunity for ACORD. Extending the NDR to include this will provide a method for generating XML constructs automatically from the ACORD data
dictionary and for keeping the XML specification automatically aligned to the
dictionary as its content grows over time. It will also become possible to incorporate
and reuse Core Components or derived XML data standards developed outside of
ACORD using compatible XML Naming and Design Rules.

Issues surrounding transmission, security, orchestration, etc. are not included in this
document. ACORD will publish separate implementation standards for these
technologies as members identify the need to support a given technology.

1.4 Compability With UN/CEFACTNDR
As exposed above, ACORD has set the goal of being compatible with NDR Standards
jointly developed by UN/CEFACT, UBL and other Standard Development Organizations,
effort in which ACORD participates.

ACORD XML schemas will be compatible as much as possible with XML schemas
generated from Core Component models, as specified in the UN/CEFACT cross-industry
NDR Specification [UNCE-NDR]. The compatibility aspects are further developed in
Chapter 2.

Consequently, ACORD is making numerous references to the UN/CEFACTNDR and has
adopted its general structure. A number of rules from that specification have been
adopted without changes, some have been adapted and additional ACORD rules
have been specified. Cross-reference to the UN/CEFACTNDR has been maintained.

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

1. MUST: this word, or the terms "REQUIRED" or "SHALL", mean that the definition is an
absolute requirement of the specification. This is tracked as a requirement.

2. MUST NOT: this phrase, or the phrase "SHALL NOT", mean that the definition is an absolute
prohibition of the specification. This is tracked as a requirement.

3. SHOULD: this word, or the adjective "RECOMMENDED", mean that there may exist valid
reasons in particular circumstances to ignore a particular item, but the full implications
must be understood and carefully weighed before choosing a different course. This is
tracked as a recommendation.

4. SHOULD NOT: this phrase, or the phrase "NOT RECOMMENDED" mean that there may
exist valid reasons in particular circumstances when the particular behavior is
acceptable or even useful, but the full implications should be understood and the case
carefully weighed before implementing any behavior described with this label. This is
tracked as a recommendation.
Example – A representation of a definition or a rule. **Examples are informative.**

Note – Explanatory information. **Notes are informative.**

[R nn] – Identification of an ACORD rule. **Rules are normative and require conformance.** In order to ensure continuity across versions of the specification, rule numbers that are deleted will not be re-issued, and any new rules will be assigned the next higher number - regardless of location in the text.

[UNR nn] Identification of a UN/CEFACT rule adopted by the ACORD NDR without modification.

[R nn] – [UNR nn] Identification of an ACORD rule, corresponding to a modified UN/CEFACT rule.

The following signs are used in the definition of rules:

- [] = optional
- <> = Variable
- | = choice

**Courier** – All words appearing in **bolded courier font** are values, objects or keywords.

Table 1.1 Prefixes and Namespaces referring to external standards, used in this specification

<table>
<thead>
<tr>
<th>Namespace Prefix</th>
<th>Namespace URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd</td>
<td><a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a></td>
<td>XML schema language</td>
</tr>
<tr>
<td>xsi</td>
<td><a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a></td>
<td>XML instance document</td>
</tr>
</tbody>
</table>

1.6 Related Documents

Related documents referenced in this specification are listed in Appendix A.
1.7 Conformance

Applications will be considered to be in full conformance with this technical specification if they comply with the content of normative sections, rules and definitions.

[UNR 1] Conformance shall be determined through adherence to the content of normative sections, rules and definitions.

[R 300] The Rules MUST be used by individuals or working groups when developing or changing an ACORD Specification.

[R 301] When extending an ACORD Specification for custom use, the Rules SHOULD be used in order to keep consistency throughout the specification and their implementations.

1.8 Governing Principles

The following principles have been applied in the development of the XML implementation of this specification.

- Remain transport and implementation independent – The XML representation of ACORD is independent of the protocols used to transport the messages between the client and server computers as well as the way it is implemented.
- Internet Use – ACORD XML messages shall be straightforwardly usable over the Internet.
- Component Reuse – The design of ACORD message types should contain as many common features as possible. The nature of e-commerce transactions is to pass along information that gets incorporated into the next transaction down the line.
- Designed for customization and reuse – ACORD messages may be customized by trading partners and if desired to be used as the basis of other implementations. When implemented as a schema, this specification can easily be customized with standard schema constructs.
- Provide for tight validation – The ACORD NDR is intended to provide guidance in building data streams without being overly restrictive or requiring excess information. Where possible, the schema should guide the implementer in what content is used rather than supporting documentation.
- Legibility – ACORD messages should be human-readable and clear.
- Simplicity – The design of ACORD messages must be as simple as possible (but no simpler).
• Standardization – The number of ways to express the same information in an ACORD message is to be kept as close to one as possible.
• Maintenance – The design of ACORD messages must facilitate maintenance.
• Legacy formats – ACORD is not responsible for catering to legacy formats;
  In developing new standards, ACORD will not be constricted by legacy XML formats and technical architectures.
  ACORD is not responsible for mapping the legacy systems with the new specifications, but should endeavor to ease migration from ACORD owned legacy standards.
• XML Technology – ACORD message design will avail itself of standard XML processing technology wherever possible (XML itself, XML Schema, XSLT, XPath, and so on). However, ACORD will be cautious about basing decisions on “standards” (foundational or vocabulary) that are works in progress.

2 RELATIONSHIP TO THE FRAMEWORK AND UNDERLYING STANDARDS

This chapter defines the linking points between the technology neutral “models” defined in the ACORD Standard Framework and the derived XML technology implementations ruled by the NDR. It presents the minimal set of assumptions on how underlying standards such SOA, Web Services, UML and the Core Component methodology are used in the Standard Framework and frame the XML implementation, so that XML Naming and Design Rules can be specified with accuracy.

2.1 Relationship to SOA and Web Services

2.1.1 Relationship to SOA

The ACORD Standard Framework is assumed to support the Service Oriented Architecture reference model (OASIS Specification) for specifying business services and associated messages.

Summary of the relevant aspects of SOA:
- The central focus of SOA is around having a task or business function done.
- A service is a mechanism to enable access to business functions using a prescribed interface
  - A service is provided by a service provider for use by service consumers that may not be known by him.
  - A service must be visible to consumers and requires the description of the interfaces that give access to the business functions
  - The description of the interface must include 2 aspects:
• **Information model**: the information that may be exchanged (structure and semantics)

• **Behavioral model**: the actions invoked against the service and the temporal behavior of the service (e.g. transactional aspects)

About the interface’s information model, or message model:

  o Message exchange is the primary mode of interaction with a service
  o There is an expectation that a message be tailored to perform the business function it supports, that its **structure be specific to the function**.

The NDR is concerned primarily with the information model and the definitions of XML messages, which are further characterized by the adherence to Web Services Standards

### 2.1.2 Relationship to Web Services

ACORD XML schemas will support the Web Services Standards (AWSP, based on W3C and OASIS Specifications).

Summary of the relevant aspects of Web Services:

- **Web Services** is one of the implementation solutions for SOA
- WSDL service description includes
  o Behavioral model (**port** concept - business function or action against the service and temporal aspects)
  o Information model (**xml message** structure - defined by a schema)
  o Service access parameters (transport and messaging **protocols**, service address etc.)

XML is the standard for messages

- WSDL has the ability to associate distinct messages to each port

  o This enables generation of specific input and output structures per business function

The NDR will support WSDL ability to specify message schema definitions at the business function granularity level. The NDR does not prescribe message granularity - the process model of the standard framework is assumed to take care of this aspect.

### 2.2 Relationship to UML

#### 2.2.1 Assumed Modeling Principles

ACORD XML schemas will be able to be derived from message and data component descriptions specified in UML. A UML subset will be used, to ensure compatibility with Core Component models.
The schema will finally be the normative thing. UML should only be used for graphical documentation, or leveraged at working group design time, recognizing that it may not represent all schema features. It does not constitute a full solution for interface implementation and deployment.

2.2.1 Definitions

Aggregation – An Aggregation is a special form of object Association that specifies a whole-part relationship between the aggregate (whole) and a component part. The aggregate and the component parts have their own independent existence and the component part may be included in other aggregates.

Composition – A form of aggregation which requires that a part instance be included in at most one composite at a time, and that the composite object is responsible for the creation and destruction of the parts. Composition may be recursive.

2.2.2 Main Assumptions

The messages will be modeled with class diagrams using exclusively:
- Aggregation relationships to business data objects
- Composition relationships to message sub-structure objects

In addition, message classes may inherit from abstract classes to assist in keeping structure consistency among all messages or messages of the same functional group.

The data object classes will be modeled using:
- Composition or aggregation relationships to other data objects

In addition, data object classes may inherit from abstract classes to assist in keeping structure consistency among data objects subtypes.

2.2.3 More Details

Message sub-structure objects will be modeled like the message class.

It must be possible to derive the order of the message parts from the message class diagram.

Class inheritance should be tempered by the following considerations:
- There are best practices for inheritance hierarchies: (a) consider how deep you keep the hierarchy (this has an impact on the schema type derivation structure), (b) consider how it provides value - stick to well understood business definitions.
In many cases, composition or aggregation of reusable aggregates or groups will allow more flexible constructs and are preferable to type hierarchies.

There are known issues around the use of UML to render XML schema constraints:

- Cardinality of a leaf property in UML (this could be solved by representing leaf elements as objects)
- Required/Optional constraint (this could be solved by representing leaf elements as objects)
- Choice/Sequence and groups in general
- Location of referenced objects in the message, when there is used in aggregations.

Out of scope for NDR:

- Whether ACORD should deliver machinable UML models
- How UML models should be kept in sync with the XML schemas.

### 2.2.2 UML Message Class Template

The general ACORD message UML template is assumed to be as follows.

**Note:** Plain lines represent compositions of objects. Dotted lines represent aggregation of objects.
2.3 Relationship to the Core Component Technical Specification

ACORD XML schemas will be compatible as much as possible with XML schemas derived from Core Component models (CCTS version 2.1 or 3.0), as specified in the UN/CEFACT NDR Specification [UN-NDR].

Specifically the ACORD schemas will be such that, starting from an actual set of Core Components or equivalent UML Class diagrams and follow the UN/CEFACT NDR rules, the generated schema will be compatible with the ACORD schemas. In other words, ACORD will support a maximum number of UN/CEFACT NDR rules so that the same set of schema features and conventions are reused. By reusing the same basic XML grammar as other business standard development organizations, development of tools and cross-industry interoperability is facilitated.

As the UML modeling assumptions exposed in the previous section are those used to describe Core Components in UML (except for inheritance), it is possible to align on the
schema derivation rules developed around Core Components to achieve the compatibility goal exposed above.

It is important to note that the NDR is only concerned with the mechanics of deriving XML schemas from Core Component or equivalent structures, not with the business content of standardized Core Components.

The only exception is that of the Core Data Types defined in the UN/CEFACT Core Component Library. These generic simple type definitions will be reused to the largest extend possible, to form an initial layer of cross-industry compatibility.
3 GENERAL XML ARCHITECTURE

3.1 Versioning and Backwards Compatibility

The ACORD NDR is designed to support Backwards Compatibility within a specific major version of a standard specification. Compatibility is defined in a manner that centers upon the XML instance documents that result from or are compliant with the use of ACORD Standards.

Please refer to Appendix E for a comprehensive discussion of Backwards Compatibility and why this approach was selected.

3.1.1 Versioning Scheme

ACORD adopts the following versioning scheme: #.#.# (the # represents a digit) in which the first digit is the Major version, the second is the Minor version, and the third is the revision number. The initial publication of the specification is 1.0.0.

3.1.2 Major Version

A Major version change indicates that the specification was modified to a degree that new compliant XML instance document may not be backward compatible with instance documents based on older Major version. If any XML instance based on such older major version ACORD Schema attempts validation against the newer version, it may experience validation errors.

Non-backward compatible changes that result in a Major version change include:
- Changing the element names, type names and attribute names
- Moving elements and attributes to another namespace
- Deleting elements or attributes in structures
- Adding mandatory elements or attributes in structures
- Changing cardinality of elements or attributes to a higher minimum value or a lower maximum value
- Changing the order of elements in structures
- Removing or changing values in enumerations

A detailed list of what is considered non-backward compatible changes can be found in Appendix E.

Major version numbers should be based on logical progressions to ensure semantic understanding of the approach and guarantee consistency in representation. Non-negative, sequentially assigned incremental integers satisfy this requirement.

[UNR 51] Every ACORD Schema major version number MUST be a sequentially assigned incremental integer greater than zero.
3.1.3 Minor Version

A Minor version change indicates that the specification is modified in a way that new compliant XML instance documents will still be backward compatible with instance documents based on an older version.

Backward compatible changes that result in a Minor version change include:
- Adding new optional structures
- Adding optional elements to structures
- Changing cardinality of elements to a lower minimum value or a higher maximum value
- Changing the data type of element’s or attribute’s simple content, opening to a larger, compatible value space.
- Adding values to enumerations

A detailed list of what is considered backward compatible changes can be found in Appendix E.

[R 52] - [UNR 52] Minor versioning MUST be limited to declaring new optional XSD constructs, extending existing XSD constructs, or refinements of an optional nature.

Minor version changes are not allowed to break compatibility with previous minor versions. Compatibility includes consistency in naming of the schema constructs to include elements, attributes, and types. UN/CEFACT minor version changes will not include renaming the schema construct.

[UNR 53] For ACORD minor version changes, the name of the schema construct MUST NOT change.

Semantic compatibility across minor versions is essential.

[UNR 54] Changes in minor versions MUST NOT break semantic compatibility with prior versions having the same major version number.

For a particular namespace, the parent major version of a schema and subsequent minor versions of a major version establish a linearly linked relationship. The first minor version must incorporate all XML constructs present in the parent major version, and each new minor version needs to incorporate all XML constructs present in the immediately preceding minor version.

[UNR 55] An ACORD minor version schema MUST incorporate all XML constructs from the immediately preceding major or minor version schema.

3.1.3.1 Adding Redundant Elements or Constructs

While the creation of redundant constructs in the standard should be avoided as much as possible, when new elements or functionality are added to the standard in subsequent versions that result in a redundancy with an existing mechanism for capturing equivalent information, the redundancy will be documented on both the new and existing item(s) while a solution to the redundancy will be explicitly listed in the documentation associated with the change.
The solution to redundancy can include a requirement that the newer mechanism must be supported beyond a specified version, controlled redundancy, deprecation, context-specific processing rules, or that both mechanisms must be supported beyond a certain version. The specifics of the solution are expected to vary depending on the redundancy being created.

3.1.3.2 Deprecation and Deletion

Minor version changes will permit deprecation and deletion of an element, structure, or code, under certain circumstances, such as following the introduction of redundant elements as explained above. Such changes will be treated as a two-step process. The item is first deprecated by making a revision change to the specification noting that the item should no longer be used and will be deleted. With a second vote to approve the deletion for a Minor version or at the next Major version the deleted item is removed from the specification.

[R 302] Under particular circumstances, such as when new elements or functionality are added to the schema that result in a redundancy with an existing mechanism for capturing equivalent information, deprecation and subsequent deletion of existing elements and structures MAY be done in a new Minor version. This MUST follow a strict approval process to guarantee that the required level of functionality is kept in the schema.

3.1.4 Revision

A Revision change indicates that the specification is not modified other than correcting errors or enhancing clarity.

A Revision change is the least significant. It must have no impact on implementations. This deals specifically with already approved items and specifications and allows ACORD to make fixes to the specification.

Revision changes include:
  - Corrections of typographical error
  - Clarifications that do not change the semantics of the elements and structures
  - Addition of ‘forgotten’, approved changes

3.2 Namespace Architecture

A namespace is a collection of names for elements, attributes and types that serve to uniquely distinguish the collection from the collection of names in another namespace. As defined in the W3C XML specification, “XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.

In practice namespaces can be used for different purposes such as:
  - Identifying distinct semantic domains, e.g. based on
Independent origin or ownership of definitions (e.g. Code lists)

Orthogonality/Independence of definitions (e.g. Processes versus data)

Modularity such as reuse of definitions across multiple semantic domains (e.g. Core data types)

Identifying major version of schemas when the XML definitions become not backward compatible

ACORD uses namespaces to manage its standards internally as well as for reusing externally defined vocabularies.

### 3.2.1 Namespace Segmentation

ACORD segments its namespaces, as required by the construction of XML messages, as shown in the following figure.
3.2.1.1 Process Domains

The namespace names used for the Messages definitions will be distinct from those used for the Business Data definitions. These namespace names can further be segmented by process domain, thus defining separate clusters of messages. The benefits of this architecture are:

The Business Process Service definitions and the corresponding messages form a vocabulary that is independent of the Business Data definitions vocabulary. Business Process Service definitions are obtained by decomposition of high level processes with a Service Orientation focus, but reuse common Business Data. Business Process domain vocabularies can evolve and be versioned independently of the Business Data vocabulary.

By further segmenting the Business Process domains, one makes it possible for those to evolve and be versioned independently of each other. Independent versioning includes selection of the Business Data Components version.

A distinct namespace for messages forces the schema to identify the message root strictly.

A distinct namespace for messages allows for custom constructions of messages for internal use. This is the spirit of WSDL message definition: build specific service messages from reusable data components.

**Note:** The actual Business Process namespace segmentation is assumed to be a business specification to be provided by the Standard Framework, based on the work done on standard process decomposition.

3.2.1.2 Data Domain

A single namespace for the Reusable Data Components aims at maximal commonality in the Insurance Data vocabulary for reuse in ACORD defined - as well as in custom defined messages. This will include the Business Data Aggregates, the Common Message Aggregates, the Business Data Types and the ACORD defined Code Enumerations (Code Content Types).

3.2.1.3 Documentation Domain

The Documentation domains will include definitions which are used only for internal documentation of the schemas. These definitions don’t impact instance documents. Two domains have been identified: the Core Data Types and the Schema Annotations.

The Core Data Types schema is the stable cross-industry, UN/CEFACT compatible, set of data types, on which the business data types are based. It is published with a distinct namespace for reference only.

The Schema Annotations schema provides definitions for ACORD specific constructs used in `xsd:annotation` items in ACORD schema modules.

In summary, the minimum number of namespaces required to build a single ACORD message schema is 3 (Process domain, Data domain and Documentation domain).
Only 2 appear in the instance documents. This may be augmented by the reuse of external code lists or external specifications.

### 3.2.2 ACORD Namespace Scheme

The ACORD namespace scheme will reflect the segmentation described above and support two additional structure levels: the schema status and major version. The figure below reflects such an approach and will be used as the basis for determining the namespace structure and rules that follow.
Figure – ACORD Namespace Scheme
### 3.2.3 Declaring Namespace

Schema composition best practices dictate that every schema module has a declared namespace.

>[UNR 38] Every ACORD defined or imported schema module MUST have a namespace declared, using the `xsd:targetNamespace` attribute.

### 3.2.4 Namespace Persistence

Once a namespace declaration is published, any change would result in an inability to validate instance documents citing the namespace. Accordingly, a change in the constructor contents of the namespace should not be allowed.

>[UNR 40] ACORD published namespace declarations MUST NOT be changed, and its contents MUST NOT be changed unless such change does not break backward compatibility.

### 3.2.5 Namespace Uniform Resource Identifiers

Namespaces must be persistent. Namespaces should be resolvable. Uniform Resource Indicators (URIs) are used for identifying a namespace. Within the URI space, options include Uniform Resource Locators (URLs) and Uniform Resource Names (URNs). URNs have an advantage in that they are persistent. URLs have an advantage in that they are resolvable.

ACORD aligns with the OAGi NDR [OAGI-NDR] recommendation that namespaces should be resolvable to a persistent location to find more information about the schema being defined. After reviewing the two URI options it was determined that

- URLs are resolvable and are as persistent as the organizations that maintain the schemas;
- URNs are not resolvable and identify a name for a given standard that is typically associated with the organization that maintains the standard;
- When the organization name changes the URN and URL change.

Therefore, URNs and URLs were determined to be equally persistent. Since URLs are resolvable, URLs are preferred.

>[R 41] - [UNR 41] UN/CEFACT namespaces MUST be defined as Uniform Resource Locators.

To ensure consistency, each ACORD namespace will have the same general structure. This namespace structure will follow the namespace scheme exposed in 1.3.2.

Following this scheme, the general structure for an ACORD namespace name will be: `http://www.acord.org/schema/<schematype>/<status>/<name>/<major>`

Where:
• `<schema type>` = a token identifying the type of schema module: data|process|documentation
• `<status>` = the status of the schema set: draft|standard
• `<name>` = the name of the schema set content (using upper camel case) with periods, spaces, or other separators removed.
• `<major>` = the major version number, sequentially assigned, first release starting with the number 1.

[UNR 42] The namespaces must have the following structure:
http://www.acord.org/schema/<schematype>/<status>/<name>/<major>.
Where: `<schema type>` = a token identifying the type of schema module: data|process|documentation;
`<status>` = the status of the schema set: draft|standard;
`<name>` = the name of the schema set content (using upper camel case) with periods, spaces, or other separators removed;
`<major>` = the major version number, sequentially assigned, first release starting with the number 1.

Example: Namespace Name at Draft Status
“http://www.acord.org/schema/data/draft/ReusableDataComponents/1”

Example: Namespace Name at Specification Status
“http://www.acord.org/schema/data/standard/ReusableDataComponents/1”

3.2.6 Namespace Constraint
To ensure consistency between namespaces and the content of the schema sets, ACORD namespaces will only contain XML constructs created and assigned by ACORD.

[UNR 44] ACORD namespace values will only be assigned to ACORD developed objects.

3.2.7 List of ACORD Standard Namespaces
The following table lists all the namespaces standardized by this specification. Standard status and major version 1 are assumed. As explained elsewhere process domain name standardization is beyond the scope of this specification.

To ensure consistency, and for standardization of namespace tokens as addressed elsewhere in this specification, all namespaces are referred to by their formal name or token value.
### 3.3 Schema Management and Versioning

#### 3.3.1 Schema Set

Schema modularity within a namespace is supported by the schema language, which allows several schema files to share the same namespace and form a “schema set”. A schema set is the set of schema modules that altogether form the XML vocabulary identified by a namespace. Per schema language rules, distinctly named objects must appear in only one module in the set.

Good practices followed by the ACORD NDR recommend:

- To encapsulate each schema set in a root schema that includes all the modules of the set (called internal schemas)
- To only import the root schema in other namespaces
- To forbid circular dependencies between internal modules, because this may cause performance problems in tools.

[R 303] Each ACORD schema set MUST have one root schema that includes all the internal schemas defined under its namespace. Inclusion may be transitive (nested), i.e. internal modules may include other internal modules.

[R 304] Internal modules MUST not form a chain of inclusions that result in circular dependencies between modules.

[R 305] When definitions residing under namespace A need to be imported in namespace B, the complete schema set corresponding to namespace A SHOULD be imported via the root schema.
3.3.2 Schema Modules Per Namespace
Namespace breakdown into schema modules can help reducing the size of the schemas and ease maintenance of the specification. ACORD schemas will be broken down as follows:

3.3.2.1 Process Domains - Message Schema Sets
Within a Process domain namespace, the following partitions in schema modules are possible:

- One schema module per namespace
- One schema module per message type hierarchy
- One schema module per message

After analysis one schema module per namespace is the preferred option. Indeed message type hierarchies may possibly vary even between minor versions and one schema per message is impractical due to the dependency on their message type hierarchy.

3.3.2.2 Data Domain - Reusable Data Components Set
After analysis of several options, it has been determined that the Reusable Data Components schema set should initially be composed of the following modules:

- The Business Data Aggregates module: will contain the aggregate definitions of the business data objects used in Insurance, as generally defined outside the context of a message.
- The Common Message Aggregates module: will contain the aggregate definitions of elements reused in the common structure of ACORD messages, e.g. those needed to add transactional information to the message, message grouping structures, or references to binary attachments.
- The Business Data Type module: will contain all the ACORD reusable simple content data types.
- The Code Lists schema module: will be composed of all ACORD defined code enumerations. This module is distinct to facilitate maintenance, identification of code lists and be sharing with other organizations. The module can be optionally reused for schema validation.

The ACORD NDR does not preclude splitting further schema modules contained in this set, provided no circular dependencies are introduced, for example shared aggregate components (Party and Address, Common Policy & Contract) could be split from unique Line of Business aggregate components. This type of decision will ultimately depend on how the ACORD Standard Framework will structure the Business Data
model. As an additional good practice, one should ensure that all type definitions derived from a root type are included in the same schema module.

### 3.3.2.3 Documentation Domain - Core Data Types and Annotations

As previously exposed the Documentation domain will contain two schema modules, each in their own namespace, the Annotations schema module and the Core Data Type schema module.

### 3.3.3 Schema and Component Versioning

#### 3.3.3.1 Schema Versioning

Namespaces provide a means for achieving consistency and harmonization between schema versions. ACORD has chosen to align namespace versioning with schema versioning. As previously exposed, the ACORD modularity approach provides for grouping of reusable schemas by a root schema. The contents of a schema set are so interrelated that proper management dictates that both versioning and namespace of all members of the set be synchronized. Schema sets have therefore been assigned to a single, versioned namespace (identifying the standard Major version).

The version number of the root schema and of all internal schemas will further be synchronized at the minor version level. Giving individual schema names that incorporate the schema set version will guarantee consistency of internal schema module cross-referencing. The Revision number will apply to individual schemas only but must be reflected in schemas that are dependent on them, including the root schema.

Distinct schema sets will be versioned independently. However the Minor version number must reflect dependencies on other schema sets.

**Example:** Independently versioned schema sets.
3.3.3.2 Schema File Names

[UNR 16] The schema module file name for modules other than code content types MUST of the form `<SchemaModuleName>_<Version>.xsd`, with periods, spaces, or other separators removed.


[R 18] - [UNR 18] In representing versioning schemes in file names, the dash (’-‘) MUST be used as separator.
3.3.3.2.1 Schema Version Indication

In instance documents Major version numbers are reflected in the namespace declaration while Minor and Revision version numbers are only reflected in the xsi:schemaLocation attribute.

Major, Minor and Revision version numbers are also declared in the version attribute in the xsd:schema element.

Just like major version numbers, minor version numbers should be based on logical progressions to ensure semantic understanding of the approach and guarantee consistency in representation. Nonnegative, sequentially assigned incremental integers satisfy this requirement.

[UNR 48] The xsd:schema version attribute MUST always be declared.

[UNR 49] The xsd:schema version attribute MUST use the following template:
<xsd:schema ... version="<major>.<minor>.<revision>"/>

[UNR 50] Every schema version namespace declaration MUST have the URI of:
http://www.acord.org/schema/<schematype>/<status>/<name>/<major>...

[R 45] - [UNR 45] The general structure for schema location MUST be a relative File Name of the form: <schematype>/<status>/<name>_<major>-<minor>-<revision>.xsd, where: <schematype> = a token identifying the type of schema module (data|process|codelist|documentation); <status> = the status of the schema (draft|standard); <name> = the name of the schema module (using upper camel case) with periods, spaces, or other separators removed; <major> = the major version number, sequentially assigned, first release starting with the number 1; <minor> = the minor version number within a major release, sequentially assigned, first release starting with the number 0; <revision> = the revision number within a minor release, sequentially assigned, first release starting with the number 0.

[UNR 46] Each xsd:schemaLocation attribute declaration MUST contain a persistent file path.

[UNR 47] Each xsd:schemaLocation attribute declaration URL MUST contain a relative file path.
3.3.3.3 Component Versioning

A key question asked by the implementer of the standard is: “How does a new schema set version affect my implementation”? When there are enough business reasons to upgrade to a new version, the implementer needs to figure out how the applications, data components and message definitions will be impacted. He would also need to understand how partners’ implementations differ from his own. Responses to this question include:

- Must be able to understand which messages in the set have had their data content changed.
- Must be able to understand which data components have had their data content changed.
- Must be able to understand the dependency of a component change on another.

These requirements cannot be met via versioning of the message schema because the version of the message schema reflects dependencies on full data schema sets, not individual components. ACORD has therefore determined that components should be versioned individually. Once messages, data components, data types etc. are versioned individually, it becomes possible via tool support to trace the cascade effect of one component change to another.

ACORD has introduced specific schema annotations for component versioning. This technique can be used whichever the way schemas are modularized.

3.3.3.3.1 Use of Schema Annotations for Component Versioning

Component versioning will use the ACORD `xsd:annotation/xsd:appinfo/acord-doc:ComponentVersionId` element. In the schema language the `xsd:annotation/xsd:appinfo` element denotes a machine-readable annotation to a schema. It provides a mechanism for application-level processors to augment schema processing with added information.

The `acord-doc:ComponentVersionId` element will reflect the value of the schema version in which the component was updated for the last time.

[R 306] All globally declared constructs in ACORD schema modules MUST be annotated with the schema version in which the component was updated for the last time, using the `xsd:annotation/xsd:appinfo/acord-doc:ComponentVersionId` element, in the format “<major>.<minor>.<revision>”.
Example: acord-doc:ComponentVersionId.

```xml
<xsd:annotation>
    
    <xsd:appinfo>
        <acord-doc:ComponentVersionId>1.0.0</acord-doc:ComponentVersionId>
    </xsd:appinfo>
</xsd:annotation>
```

3.3.3.4 Indicating Schema Version And Message Component Version In Message Instance

The `xsd:schemaLocation` attribute will reflect the schema version number via the schema name and a separate Component Version Identifier element will indicate the component version.

[R 307] A Version Identifier element MUST be defined as a required child of each ACORD message root element to indicate the message component version in the message instance.

3.3.3.5 Schema Slicing/Slimming

The namespaces and schema set segmentation focus primarily on specification modularity, not on operational optimization. Schema slicing tools may still be needed to help operational optimization, e.g. for providing a user with content limited to a given business message (removal of noise). Slicing may only change the content and organization of the schema modules within namespaces, not change the namespace structure, as it otherwise would produce incompatible message instances.
4 Schema Design And Generation

4.1 Schema Design Style

4.1.1 General Principles

[R 308] General schema design SHOULD be supported by standard XML tools. In general, new schema features should be tested in the Java and Microsoft environments to make sure there are tools that support the features being implemented.

[R 309] When there is a choice between a design that provides tight validation with a schema and one that doesn't; you SHOULD choose the option that provides tight validation. A tighter design provides implementation guidance via the schema, while in a looser design additional implementation guidance is required to make the design useful.

4.1.2 Reusability of Schema Objects

One of the key objectives that had to be pursued for determining ACORD schema design rules has been reusability of schema objects. Reusability requirements include:

Type reuse within schemas: aim at maximal reuse of both simple and complex types in element definitions, in support of semantic clarity and implementation simplicity.

Type reuse in processing software: post schema-validation info set (PSVI) capabilities are beginning to emerge that support this approach, such as “data-binding” software that compiles schema into ready-to-use object classes and is capable of manipulating XML data based on their types.

Element reuse: element reuse is important within schemas as well as in derived technologies like WSDL; element reuse allows for binding semantically independent objects in documents and messages.

Ability to customize ACORD schemas, i.e. to easily derive custom XML structures from ACORD standard schema structures; customization of ACORD schemas requires both type and element reuse.

Reusability of elements and types should be semantically meaningful. Therefore the closest alignment should be sought between schemas and the UML or Core Component models which they reflect.

Like UN/CEFACT, ACORD deliberated adopting a type based approach and an element and type based approach.
The type based approach (aka Venetian Blind schema design pattern) indicates all elements other than the root element as defined locally using globally defined types. This approach is suitable for type reuse in processing software which provide data-binding facilities. The most significant issue with the type based approach is the risk of developing an inconsistent element vocabulary where elements are declared locally and allowed to be reused without regard to semantic clarity and consistency across types. This risk can be managed by carefully controlling the creation of element names with fully defined semantic clarity in modeling tools that provide input to schema generation.

A purely type based approach does, however, limit the ability to reuse elements. Reusing elements is necessary in technologies such as Web Services Description Language (WSDL), for schema customization, and simply for accurate schema design to render objects that are semantically independent.

The element and type based approach (aka Garden of Eden schema design pattern) indicates all elements as defined globally with globally defined types. The most significant issue with the element and type based approach is the assumption that all elements can be reused, including leaf elements. However for leaf elements to be reusable they must have a global name, business definition and data type. This unduly overloads naming and limits data type restrictions applicable in the context where the elements reside. For example:

- `<Description>` can only be given specific length and meaning if given specific names, e.g. AbbreviatedDescription, FullDescription, etc.
- `<Height>` could have maximum value constraints that differ from object to object (building height, door height ...), which would require distinct element names.

These overloaded names result in leaf elements that may not be actually reused (since they have specific definitions). It should be noted however that, even if the type based approach is used, inheritance associations (rendered in schema by derivation by extension) result in ‘freezing’ element definitions by making them common among the class subtypes. Therefore, if not correctly designed, inheritance associations may make the `xsd:complexType` structure less flexible.

### 4.1.3 Hybrid Design Pattern

In accordance to UN/CEFACT, ACORD has thus decided to implement what is known as a “hybrid approach” as this provides benefits over a purely type based approach. Most significantly it increases reusability of library content both at the modeling and schema level.

The key principles of the “hybrid approach” are:

- All classes are declared as a `xsd:complexType`.
- All simple properties (aka attributes) of a class are declared as a local `xsd:element` within a `xsd:complexType`.
- Classes that are associated through compositions are locally declared as a `xsd:element` within a `xsd:complexType`
Classes that are not associated through compositions are globally declared as a `xsd:element`. (Global elements are used to represent reusable constructs that have sufficient semantics independent of the context in which they are used).

Inheritance associations - that indicate classes that inherit the source class - are represented using `xsd:complexType` derivation by extension.

The rules pertaining to the ‘hybrid approach’ are detailed in Chapter X - Data schemas.

### 4.1.4 XML Referencing

#### 4.1.4.1 Defining And Referencing Objects in a Document Instance

##### 4.1.4.1.1 Structural Referencing

Logical relationships between XML aggregate elements in an instance document can be represented either by nesting elements hierarchically or by creating links between separate elements.

A key decision factor in choosing which approach to use is determining how many relationships the element instance is part of in the structure of a message. For example, an address or party instance typically has many relationships with other elements in an insurance message; but a car has only a single engine associated with it. When the structure of a message or the underlying business data model require enforcing a relationship between two elements the relationship must be enforced in the schema using adequate referencing constraints.

##### 4.1.4.1.2 Dynamic Referencing

Any element composing a message is potentially the target of a reference for the purpose of building dynamic relationships between elements within the message and may require specifying an optional identifier property to this element. Such identifiers are typically used to build short XPath expressions pointing to the XML element. An important use case is identification of faulty elements for error reporting. The only requirement at processing time is that the identifiers are found unique across all the elements of the document fragment in which these XPath expressions apply.

#### 4.1.4.2 Schema Validation

##### 4.1.4.2.1 Validation Requirements

ACORD has opted for the following validation requirements:

For structural references, the level of validation performed by the schema definition of a message should be as strong as if the referenced element would have been defined as a nested child of the element that references it. Thus, the schema must strictly enforce identity constraints, i.e.: (1) check uniqueness of the identifiers
of the referenced elements (2) check that the references match the identifiers of the corresponding referenced elements.

For dynamic referencing this level of validation is not required. Since dynamic referencing is only used for ancillary purposes, it is not deemed essential to enforce uniqueness of identifiers in the schema when they are not involved in structural referencing. Uniqueness of such identifiers should be granted by use of adequate algorithms for the generation of the identifiers (e.g. UUIDs). This will avoid unnecessary complexity of the identity constraints.

4.1.4.2.2 Schema Identity Constraints

For schema validation of structural references, ACORD will exclusively use XML Schema \texttt{xsd:key}, \texttt{xsd:keyref} or \texttt{xsd:unique} identity constraints to specify a relationship between objects within a XML document. These schema constraints have the following characteristics that make them preferable to the ID/IDRef technique.

The keys and relationships between objects are strongly typed. They are declared explicitly in the schema. Each relationship is distinctly defined and specifies exactly which object has a key, what is the key, which other objects can link to this object and through which element or attribute. You can prevent an object to point to any other object that has an identifier attribute, as it is the case with the ID/IDREF method.

The scope of key uniqueness is precisely defined among one or several objects within a particular instance of an XML element. It is not more necessary to ensure uniqueness of id attributes across the whole XML document, which could be e.g. a batch of messages.

The elements or attributes used as keys or key references can be of any data type, not only ID or IDRef (implying the NMTOKEN format). This allows any element or attribute to be used for linking.

\cite{R67} - \cite{UNR67} The \texttt{xsd:key}, \texttt{xsd:keyref} or \texttt{xsd:unique} identity constraints as defined by the XML Schema Specification \textbf{MUST} be used when it is necessary to validate a structural reference from an object to another.

\cite{UNR66} \texttt{xsd:ID/xsd:IDREF} \textbf{MUST NOT} be used for information association.

4.1.4.2.3 Usage of Key/KeyRef for Structural Referencing

Usage of key-keyref for structural referencing will be limited to two cases, in the frame of this specification.

\cite{R312} Usage of key-keyref for structural referencing \textbf{MUST} be limited to two cases:

(1) When the structure of a message or the underlying business data model require enforcing a relationship between two elements.

(2) For use in Core Data type as an alternative to element attributes, e.g. for code list metadata.
Please refer to Chapter 7 - XML schema modules for detailed rules on how XML referencing must be implemented.

4.2 XML Naming Rules

4.2.1 General

[R 313] Every globally declared schema object within an ACORD schema set (one namespace) MUST have a unique name; case differences do not qualify as a unique name.

Note: Objects as used here include all XML objects that are required by this specification to be globally named, e.g. global elements, global attributes, groups and types. Code values have their own naming conventions exposed in Section 4.4.2.

Even though XML allows for any of these to use identical names or have names that only differ in case; ACORD requires a unique name to be assigned to help with maintenance and support of its specifications.

Example

<table>
<thead>
<tr>
<th>VALID</th>
<th>NOT VALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element: Policy, type: Policy_Type, attribute: policyReference</td>
<td>Element: Policy, type: Policy, attribute: policy</td>
</tr>
</tbody>
</table>

[R 16] - [UNR 16] ACORD global names MUST be composed of words in the English language, using the primary English spellings provided in the Oxford English Dictionary. All words in the name of an object MUST be in the English language and the spelling MUST be based on the primary English spellings in the Oxford English Dictionary.

[R 10] - [UNR10] Of the characters allowed in an XML name, an ACORD global name MUST only use upper and lowercase letters. A though Z in addition “_” (underscore) MUST be used for schema type and group name suffixes (e.g. “_Type”, “_List”, “_Choice”, “_Sequence”). Accordingly XML names in ACORD MUST begin with a letter.

Note: This rule supports the principle that names must be composed of English words only. There is one exception to this rule for acronyms, which may contain other
characters as exposed in Section 4.2.2. Code values have their own naming
conventions exposed in Section 4.4.2.

Example

VALID

<table>
<thead>
<tr>
<th>Element: AccountNumberId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Policy_Type</td>
</tr>
</tbody>
</table>

NOT VALID

<table>
<thead>
<tr>
<th>Element: AccountNumberId&amp;Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Policy-Type</td>
</tr>
</tbody>
</table>

[R 11] - [UNR 11] XML element, attribute and type names MUST be
constructed from the names given in the UML model or the Core Component
from which they are derived. They MUST NOT include periods, spaces, or other
separators; or characters not allowed by W3C XML 1.0 for XML names.

Construction of XML names from Core Components names is fully specified in the
UN/CEFACT NDR document [UN-NDR].

[R 314] When no UML or Core Component source names are available, XML
names SHOULD follow the same general rules as those adopted for Core
Components, i.e. those of the ISO 11179-5 standard (Standardization of data
elements - Naming and identification principles). Applicable rules are presented
below.

Name parts normalized by ISO consist of discrete terms. These are: object class
terms, property terms, representation terms, and qualifier terms. The general
structure of a data element name is as follows: <object class term>, <property
term>, <representation term>.

Example - ISO name

<table>
<thead>
<tr>
<th>Roof Type Code:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roof is the object class term, Type the property term and Code the representation term</td>
</tr>
</tbody>
</table>

Previous Policy Effective Date:

<table>
<thead>
<tr>
<th>Policy Effective Date:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy is the object class term, Date is the combined property term and representation term</td>
</tr>
</tbody>
</table>

NOT VALID

Type Roof Code
Qualifier terms may be placed before object class terms and property terms if necessary to distinguish one data element concept from another. A qualifier here can be an adjective or a word that results in a compound word.

**Example - ISO name**

<table>
<thead>
<tr>
<th>Previous Policy Effective Date:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous is the qualifier term of the object Policy, Effective is the qualifier term of the property Date</td>
</tr>
</tbody>
</table>

**NOT VALID**

<table>
<thead>
<tr>
<th>Effective Previous Policy Date</th>
</tr>
</thead>
</table>

The name of a complex element derived from an ISO name corresponds to the name of the object class it represents; therefore it MUST be composed of the object class term optionally preceded by a qualifier term, with spaces or other separators removed.

**Example**

<table>
<thead>
<tr>
<th>PreviousPolicy</th>
</tr>
</thead>
</table>

The name of a simple element or an attribute derived from an ISO name corresponds to the name of the property it represents; there are several sub-rules for XML name derivation:

- The element or attribute name MUST start with the property term optionally preceded by a qualifier term, with spaces or other separators removed.

- The element or attribute name MUST end with the representation term depending on its core data type; this dependency is presented in the core data type section.

- The element or attribute name SHOULD NOT include the object class term when it is a child of a complex element representing the object class itself.

**Example - Xpath expressions**

**Right**

<table>
<thead>
<tr>
<th>PreviousPolicy/EffectiveDate</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Building/RoofTypeCode</th>
</tr>
</thead>
</table>

**Overloaded element name**

<table>
<thead>
<tr>
<th>PreviousPolicy/PolicyEffectiveDate</th>
</tr>
</thead>
</table>
When creating an XML name, you SHOULD remove prepositions such as "of", "and", "or", and "the" from names unless it is critical to make sense of the name. An object name SHOULD only contain verbs, nouns, and adjectives unless it adds clarity to the name.

Example

Right
AsOfDate

Unclear
AsDate

XML names MUST be in singular form unless the concept itself is plural.

Example

VALID
ItemCount
GoodsQuantity

NOT VALID
ItemsCount
GoodQuantity

XML names MUST NOT use acronyms, abbreviations, or other word truncations, except those included in the ACORD controlled vocabulary as exposed in Section 4.2.2.

[77] - [79] Lower camel case (LCC) MUST be used for naming attributes. Lower Camel Case capitalizes the first character of each word except the first word and compounds the name.

Example - Attribute
unitCode

[78] - [79] Upper camel case (UCC) MUST be used for naming elements and types and groups. Upper Camel Case capitalizes the first character of each word and compounds the name.

Example - Element
VALID
RoofTypeCode
[R 317] When describing a general role or class, you MUST avoid naming it with a particular "instance" of this role or class.

If SSN (Social Security Number) is being used to capture a general tax identifier, a more appropriate name might be PersonTaxIdentifier. In the description of PersonTaxIdentifier, you should include a reference to SSN and other values that might be used. However, if SSN is used as the official identifier for the US provided retirement program, SSN cannot be substituted by something less specific. In the UK, National Insurance Number (NIN) would be the equivalent to a SSN and it would be the appropriate value to use in a standard used in the UK.

[R 318] Product names and trademarks MAY be used when that specific product is being referenced. A product name MUST not be used when it is applying to a class of products.

Example
If the number of KLEENEX® brand facial tissues boxes from Kimberly-Clark is required, then use <KleenexBoxCount>, but if the requirement is for the number of "tissue" boxes, use <TissueBoxCount>.

4.2.2 Rules for Abbreviations and Acronyms

An acronym is a word formed from the initial letters of a name; while abbreviations are a shortened form of a word. Typically, acronyms are associated with the names of organizations or standard business or technical terms; while an abbreviation can be used to shorten any word.

4.2.2.1 Abbreviations
As a general rule, there are no abbreviations allowed in ACORD XML names.

Abbreviations are strictly limited to being used as a suffix to name the representation term of an element or attribute, as shown in the Core Data Type Section.
Abbreviations of words in an XML name MUST NOT be used. Only the values listed in the Core Data Type Section are allowed and only as suffix as indicated depending upon the type of object and its data type.

4.2.2.2 Acronyms

Appendix A lists the only acronyms that can be used in an ACORD XML name. In general, items are included if they are considered to be commonly used and widely understood and are "standard" to the business of insurance or technical terminology.

[R 13] - [UNR 13] If an acronym is defined in Appendix A, that acronym MUST always be used in a name.

Example

<table>
<thead>
<tr>
<th>VALID</th>
<th>NOT VALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPAmount</td>
<td>PersonalInjuryProtectionAmount</td>
</tr>
</tbody>
</table>


Example

<table>
<thead>
<tr>
<th>VALID</th>
<th>NOT VALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>isoCode</td>
<td>iSOCode</td>
</tr>
<tr>
<td>countryISOCode</td>
<td>countryISOCode</td>
</tr>
</tbody>
</table>


Example

<table>
<thead>
<tr>
<th>VALID</th>
<th>NOT VALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISOCode</td>
<td>IsoCode</td>
</tr>
<tr>
<td>CountryISOCode</td>
<td>CountryISOCode</td>
</tr>
</tbody>
</table>
When an acronym is used in the name of an object, its description MUST indicate the exact meaning of the acronym.

The acronym must be spelled out in the description and then followed with the acronym in parentheses. For example: If SSN is used as a name, the description would indicate: “Social Security Number (SSN) captures the identifier used in the US to manage an individual's government retirement account”.

### 4.2.3 Code List And Value Names

Please refer to Section 4.4 for Code List and Value naming rules.

### 4.2.4 Data Type And Group Names

Data types and groups must conform to the general naming rules as set forth in previous sections and the specific rules found in this section.

Data types and groups have particular naming conventions presented for information in the table below. Please refer to Chapter 7 for normative application of these conventions.

#### Table - Data Types and Groups Naming Rules

<table>
<thead>
<tr>
<th>Data Type or Object Type</th>
<th>XML Data Type Name Rules</th>
<th>XML Tag Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple properties of object classes – Simple Types or Complex Types with Simple Content</td>
<td>Simple Types or Complex Types with Simple Content MUST follow Core and Business Data Type naming conventions.</td>
<td>_Type or usage of xsd built-in type</td>
</tr>
<tr>
<td>Role classes - Complex Types with Simple Content</td>
<td>The name of the xsd:complexType MUST be composed of the name of the referenced element followed by the suffix 'Role_Type' and the name of the local attribute MUST be composed of the referenced element, followed by the suffix 'Reference'</td>
<td>Role_Type</td>
</tr>
<tr>
<td>Object classes – Complex Types with Complex Content</td>
<td>When data type is instantiated by a single element, use: &lt;element name&gt; + '_Type', When the data type is abstract use: &lt;descriptive name&gt; + '_Type'.</td>
<td>_Type</td>
</tr>
<tr>
<td>Code Types</td>
<td>The Code Types are the data types associated to a particular Code List. For Code Type names, use: &lt;code list name&gt; + '_Type'</td>
<td>_Type</td>
</tr>
<tr>
<td>Code Content Types</td>
<td>The Code Content Types are the simple types used to specify Code List value facets and enumerations. For Code Content Type names, use: &lt;code list name&gt; + 'Content_Type'</td>
<td>Content_Type</td>
</tr>
<tr>
<td>Element Groups</td>
<td>Names for groups SHOULD be based upon the description of what they are grouping and why, followed by the following suffix: sequence – '_Sequence' choice or Exclusive OR – '_Choice' Inclusive OR – '_Or' (combination of sequence and choice)</td>
<td>_Sequence _Choice _Or</td>
</tr>
<tr>
<td>Attribute Groups</td>
<td>Names for groups SHOULD be based upon the description of what they are grouping and why, followed by the suffix 'Attribute_Sequence'</td>
<td>Attribute_Sequence</td>
</tr>
</tbody>
</table>
4.3 Core Data Types

4.3.1 Compatibility With Core Components

As exposed in Section 2.3, ACORD intends to be compliant with the Core Data Types defined in the UN/CEFACT Core Component Library. These generic simple type definitions will be reused to the largest extend possible, to form an initial layer of cross-industry compatibility because:

- The Data Type library defined in CCTS is a key base for the semantic definitions of simple elements, which we have all benefits to reuse.
- If we have to reuse XML components from other standards, we would like to have the same set of data types for low level compatibility of our implementations (e.g. an amount, percent, code should be defined the same in XML schemas).

By reusing the Core Data Types defined in the UN/CEFACT Core Component Library we ensure that all standard organizations start from the same templates before defining Industry specific business data types by applying further restrictions.

4.3.1.1 Representation Terms

Compliance with the Core Components Technology (CCTS) requires looking at the 'Representation Term' concept. This concept is used in CCTS data element naming as the suffix of a name (e.g. "Location. Description. Text").

A Representation Term for a simple element serves the purpose of specifying the content type of that element (e.g. "Text", "Numeric" etc.) and is a part of its semantic definition.

The Representation Terms for simple elements form a controlled vocabulary, i.e. a closed list of preferred terms for the main content type categories; this enhances semantic clarity in a cross-industry and international context.

4.3.1.2 Core and Business Data Types

In CCTS, each Representation Term is linked to one Core Data Type, which defines the details of the content model and the facets by which it can be restricted. The content model can include metadata attributes (e.g. @currencyCode for Amount types).

Core Data Types only represent the main content type categories that can be used in any business domain. This list is intended to be very general and stable.

Core Data Types are also designed to be the templates from which any other specific data type needed by a business domain can and must be derived.

Such derived specific data type for a business domain is called Business Data Types. A Business Data Type must always be a restriction of the content model of a Core Data Type or another Business Data Type. Restrictions can be done:
by constraining the data simple content using the facets defined for the source Core Data Type (e.g. text max length);
so that the set of metadata attributes defined for the source Core Data Type is reduced.

4.3.1.3 UN/CEFACT NDR Naming Conventions

The UN/CEFACT NDR translates Representation Terms into XML element name suffices, so that this part of semantic definition is not lost in the XML vocabulary.

The strict link between a XML element name suffix and the source Core Data Type is an element of precision and quality of naming because it gives information to the business user about the value space of the element and avoids mapping mistakes.

Another important UN/CEFACT NDR rule applies to Business Data Types naming. Their names must be constructed from the name of the source Core Data Type or Business Data Type from which they are derived, with a qualifier term prefix so that the name of the source construct remains visible, following the general CCTS naming conventions.

4.3.2 Application to the ACORD NDR

ACORD adapts slightly the UN/CEFACT NDR rules related to Core Data Types as follows.

4.3.2.1 CoreData Types, Representation Terms, Element Name Suffices

ACORD relaxes the UN/CEFACT rule by which each CCTS representation term and element name suffix is mapped to one and only one Core Data Type. Indeed, a number of the CCTS Core Data Types are equivalent for what their content model and semantic meaning are concerned, as this shows particularly for numeric data types. For such equivalent Core Data Types, ACORD has selected a single preferred Data Type and allowed for several element name suffices, while still keeping the mapping to CCTS representation terms. So a given suffix must point to only one core data type but a core data type can be represented using several suffices.

4.3.2.2 Use of W3C Schema Simple Data Types

Following the UN/CEFACT NDR, Core Data Types are all named types in a dedicated, specific, namespace. Core Data Type names follow the controlled list of Representation Terms. ACORD relaxes this rule by permitting a certain number of xsd built-in simple types to serve as Core Data Types when an ACORD named type would add no value and would have the same content model and semantic meaning. This eases interoperability and clarity of the data type definition by adhering to the original schema definition.

[R 321] The xsd built-in simple type names MUST be used unless they are modified by the ACORD NDR Specification. These types will be used as specified with no artificial constraints (i.e. xsd:boolean will be used as defined and its values will not be limited to true and false. ACORD will allow 0 and 1 to be used as well).
The list of usable xsd built-in simple type is strictly limited to match the controlled vocabulary of Representation terms.

[R 322] Other xsd built-in simple types than those defined for use as Core Data Types by the ACORD NDR MUST NOT be used.

4.3.2.3 Core Data Type for Code Lists

ACORD modifies the UN/CEFACT Core Data Type as follows:

4.3.2.3.1 Metadata at Code Element Instance Level – Attributes

ACORD restricts the metadata attributes for each code element instance to:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Cardinality</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>@name</td>
<td>Optional</td>
<td>The textual equivalent of the code value</td>
</tr>
<tr>
<td>@xml:lang</td>
<td>Optional</td>
<td>The identifier of the language used in the code name</td>
</tr>
<tr>
<td>@codeListReference</td>
<td>Optional</td>
<td>The XML reference to a Code List descriptor, i.e. the CodeList element.</td>
</tr>
</tbody>
</table>

Example:

```
<LossCauseCode name="Negligence" xml:lang="en" codeListReference="Code1">Negligence</LossCauseCode>
```

4.3.2.3.2 Metadata at Code List Level – Descriptor Aggregate

ACORD specifies the code list metadata in a separate descriptor aggregate, the CodeList element with the following content aligned with the corresponding UN/CEFACT metadata attributes

CodeList descriptor aggregate:

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Data Type</th>
<th>Cardinality</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>@key</td>
<td>SimpleIdentifier_Type</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>ListId</td>
<td>SimpleIdentifier_Type</td>
<td>Optional</td>
<td>The identification of a list of codes</td>
</tr>
</tbody>
</table>

Example:
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListName</td>
<td>LongText_Type</td>
<td>Optional</td>
<td>The name of a list of codes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISO language code (external)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Loss Cause Code (ACORD)</td>
</tr>
<tr>
<td>ListVersionId</td>
<td>SimpleIdentifier_Type</td>
<td>Optional</td>
<td>The Version of the code list</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1988 (external)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.0.0 (ACORD)</td>
</tr>
<tr>
<td>ListAgencyCode</td>
<td>ACORDClosedCode_Type</td>
<td>Optional</td>
<td>An agency that maintains one or more code lists. The code value must be in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the specified ACORD Closed List.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ACORD</td>
</tr>
<tr>
<td>ListAgencyName</td>
<td>LongText_Type</td>
<td>Optional</td>
<td>The name of the agency that maintains the code list</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>International Standard organization</td>
</tr>
<tr>
<td>ListURI</td>
<td>xsd:anyURI</td>
<td>Optional</td>
<td>A URI that identifies the code list. Can be used to identify the URL of a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>source that defines the set of currently approved permitted values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reference IETF RFC 3986</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td>ListSchemeURI</td>
<td>xsd:anyURI</td>
<td>Optional</td>
<td>A URI that identifies the code list scheme</td>
</tr>
</tbody>
</table>

**Example:**

```
<CodeList key="Code1">
  <ListId>LossCauseCode</ListId>
</CodeList>
```
4.3.3 ACORD Data Type Mapping to Core Component Data Types

ACORD has opted to map to the UN/CEFACT Core Data Types as described in the following table.

A small number of Core Data Types as well as Representation Terms required for use in ACORD were missing and are in the process of being submitted to UN/CEFACT. ACORD extensions to the current UN/CEFACT Core Data Types appear in italic. Some of the UN/CEFACT Core Data Types cannot directly be reused in ACORD but must be restricted to be usable. These Core Data Types are marked as abstract types.

Permitted XML element name suffices in relation to the Core Data Types are shown in the table.

An initial list of ACORD derived Business Data Type is shown in the table. This list is meant to grow as business requirements grow in the future.

Detailed definitions of the ACORD Core Data Types can be found in Appendix B.

Table - ACORD Data type mapping to Core Component Data Types

<table>
<thead>
<tr>
<th>CCTS Representation term</th>
<th>UN NDR XML suffix</th>
<th>CCTS Core Data type (version 3.0)</th>
<th>22 types</th>
<th>ACORD XML Core Data Type</th>
<th>ACORD XML suffix</th>
<th>ACORD derived XML Business Data Type (initial list)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>Numeric</td>
<td>NumericType</td>
<td></td>
<td>Numeric_Type</td>
<td>Numeric</td>
<td>IntegerNumeric_Type, UnsignedIntegerNumeric_Type</td>
</tr>
<tr>
<td>Value</td>
<td>Value</td>
<td>ValueType</td>
<td></td>
<td>Value</td>
<td></td>
<td>DegreeLatitudeNumeric_Type</td>
</tr>
<tr>
<td>Ratio</td>
<td>Ratio</td>
<td>RatioType</td>
<td></td>
<td>Ratio</td>
<td></td>
<td>DegreeLongitudeNumeric_Type</td>
</tr>
<tr>
<td>CCTS Representation term</td>
<td>UN NDR XML suffix</td>
<td>CCTS Core Data type (version 3.0)</td>
<td>ACORD XML Core Data Type</td>
<td>ACORD XML suffix</td>
<td>ACORD derived XML Business Data Type (initial list)</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------</td>
<td>----------------------------------</td>
<td>--------------------------</td>
<td>-----------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Percent</td>
<td>Percent</td>
<td>Percent_Type</td>
<td>Percent_Type</td>
<td>Percent</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>TextType</td>
<td>Text_Type (abstract)</td>
<td>-</td>
<td>InfiniteText_Type, LongText_Type, InfiniteLanguageText_Type, LongLanguageText_Type; PhoneNumberText_Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>NameType</td>
<td>Name</td>
<td>Name</td>
<td>-</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Date Time</td>
<td>DateTime</td>
<td>DateTimeType</td>
<td>xsd:dateTime</td>
<td>DateTime</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>xsd:gYear</td>
<td>Year</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>xsd:gYearMonth</td>
<td>YearMonth</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>xsd:gMonth</td>
<td>Month</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>xsd:gMonthDay</td>
<td>MonthDay</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>xsd:gDay</td>
<td>Day</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>DateType</td>
<td>DateType</td>
<td>xsd:date</td>
<td>Date</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>TimeType</td>
<td>TimeType</td>
<td>xsd:time</td>
<td>Time</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Duration</td>
<td>DurationType</td>
<td>DurationType</td>
<td>xsd:duration</td>
<td>Duration</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Indicator</td>
<td>IndicatorType</td>
<td>IndicatorType</td>
<td>xsd:boolean</td>
<td>Indicator</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>CodeType</td>
<td>CodeType</td>
<td>Code_Type</td>
<td>Code</td>
<td>ACORDClosedCode_Type, ACORDOpenCode_Type</td>
<td></td>
</tr>
<tr>
<td>Identifier</td>
<td>IDType</td>
<td>IdentifierType (abstract)</td>
<td>Id</td>
<td>AssignedIdentifier_Type, SimpleIdentifier_Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UUID_Type</td>
<td>UUID</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>xsd:anyURI</td>
<td>URI</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>URL</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>URN</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
4.3.4 Indicators, Responses, Underwriting Questions

Elements of type indicator (or xsd:boolean) are used to represent simple Yes/No constructs. An indicator type is typically created when:

All that is needed is an indication that a particular thing exists.

There is a simple check box question on the ACORD form; "Current residence is: Owned? - Rented?"

[R 323] When capturing data that is a simple yes/no or true/false, the xsd:boolean data type MUST be used.

Underwriting questions generally have a more complex meaning and require more response details than Yes/No. When this is the case, a code element associated to the proper Code List must be used in order to reflect the required answer values, such as "not answered" or "not applicable".
[R 324] When capturing data that requires more than a simple yes/no, you MUST create a code element and associate it to the proper Code List in order to reflect the required answer code values.

### 4.4 Code List Management

#### 4.4.1 What is a Code List

In a technology neutral format, a Code List can be seen as a tabular structure. The following example shows a Payment Means table:

**Example – Payment Means**

<table>
<thead>
<tr>
<th>XML Code</th>
<th>Name</th>
<th>Description</th>
<th>Sublist identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmericanExpress</td>
<td>American Express</td>
<td>American Express Credit Card Payment</td>
<td>Credit Card</td>
</tr>
<tr>
<td>Cash</td>
<td>Cash</td>
<td>Cash Payment</td>
<td>Non-Credit Card</td>
</tr>
<tr>
<td>DinersClub</td>
<td>Diner's Club</td>
<td>Diner's Club Credit Card Payment</td>
<td>Credit Card</td>
</tr>
<tr>
<td>EFT</td>
<td>Electronic Funds Transfer</td>
<td>Electronic Funds Transfer Payment</td>
<td>Non-Credit Card</td>
</tr>
<tr>
<td>PayrollDeduction</td>
<td>Payroll Deduction</td>
<td>Payroll Deduction Payment</td>
<td>Non-Credit Card</td>
</tr>
<tr>
<td>Visa</td>
<td>Visa</td>
<td>Visa Credit Card Payment</td>
<td>Credit Card</td>
</tr>
</tbody>
</table>

Each row of the table represents a code entry and each column represents a property of the code entry. In this example the first column is used as a unique code identifier, the other columns are descriptive properties and their number is not limited, depending on the descriptive needs of a specific Code List.

Code Lists are very common in the ACORD Specifications. Some of these Code Lists are developed and maintained by ACORD and others are referenced from their source, either from other standards bodies, or from companies and their trading partners. As such, there are two separate sets of rules for their design and use: those for the ACORD owned lists that it can control and those for code lists owned by others, which are out of ACORD’s control and have fewer restrictions on their design.

Below is the assumed common set of ACORD owned code table properties:

- **Code value**: unique code entry identifier
- **Code name**: display name or concise clear text description
- **Code description**: full semantic definition including usage description
- **Deprecation indicator**: indicator of whether the code value has been deprecated but not removed from the Code List
- **Creation date**: date of creation of the code entry
- **Sublist identifier**: identifier of a sublist (see definition in next sub-section)

The list of unique code identifiers aka ‘Code Values’ forms the set of values that can be given to an XML element defined as a code.
Any Code List MUST represent a single semantic concept so that it can be associated to an XML element that has the same semantic meaning.

Being technology neutral, nothing would prevent a code table from containing more than one identifier column, if this fits the needs of other technologies than XML. It might be necessary for example to maintain a parallel set of identifiers for XML, EDI or Form documents. For the purpose of the XML NDR, it is assumed that only one code identifier property is applicable in the construction of XML documents.

Any Code List MUST have one single Code Value (code identifier) property for usage in ACORD XML documents.

Code names can be reflected in instance documents and, as such, should always be documented in code tables.

Every code entry SHOULD have a Name property. This must be a clear and concise text description of the code value represented. This description MUST provide enough detail to make it easy to differentiate between each code value in the list.

All non-ACORD codes lists SHOULD be defined with enough information that they can be obtained from their source or found on the Internet. When possible, for each non-ACORD code list, provide the following information: Source, Title, and Version.

When there are multiple forms of the code values (numeric, two or three letter code values, etc.), ACORD MUST specify which of these options should be used in an ACORD document.

4.4.1.1 Sublist Identifier Property

In ACORD owned Code Lists, a sublist identifier may be used as a documentation technique for logically grouping code entries within a code list.

In the Payment Means code list example above, entries are presented in two groups with identifiers ‘Credit Card’ and ‘Non-Credit Card’.

This technique is used to organize long lists in documentation. It is not intended to generate code list subset schemas for instance validation.

A sublist identifier provides a way to group or classify codes within a list following one or several hierarchies that could have more than one level, e.g. main class, subclass, code item, for documentation and presentation.
4.4.2 Code List Naming Rules

4.4.2.1 Applicable General Rules

The following general rules are applicable to Code Lists:

R 9 and R316

An ACORD Code List is a list whose values are maintained by ACORD.

[R 330] ACORD Code Lists MUST be given a globally unique name so that their type definition and content can be managed in the schema. An ACORD Code List Name SHOULD be equal to the name of the element to which it is associated. According to the Core Data Type list, an ACORD Code List Name MUST end with the ‘Code’ suffix.

Example:

**BuildingExposureCode**: A code identifying the proximity of the exposure from the building

**DriverTypeCode**: A code that identifies the driving status of the driver

An External Code List is a list used in an ACORD Specification but whose values are maintained by another organization.

[R 331] All External Code Lists MUST be given a globally unique name so that their type definition can be given an explicit name in the schema. According to the Core Data Type list, an External Code List Name MUST end with the ‘Code’ suffix.

Examples:

**ISOCountryCode, ISOCurrencyCode**

**Note:**

The unique name allows members and trading partners to incorporate those lists (if they wish) or to override them with another code list while keeping the element name. So, as an example, an element using the ISO country codes will have a name like **CountryCode**, and its code list will be **ISOCountryCode**.

4.4.2.2 Code Values

[R 332] Code values and names in all ACORD-owned code lists MUST follow the naming and design rules specified in this section.
ACORD code values MUST only use upper and lower case letters A through Z, digits 0 through 9, or the underscore "_" sign used as leading character.

A numeric value as a code SHOULD be avoided in an ACORD code list. If a numeric code value is the best possible value, the name MUST begin with a letter or an underscore "_". However a letter SHOULD be used in this situation unless the addition makes the code less clear.

Example

VALID

401K is a well known US retirement program, as such it is a good value, and the code in an ACORD list has to be: _401K.

In a list of personal auto insurance, deductible of 100, 500 and 1000 dollars make sense as a number and would be code values: USD100, USD500, USD1000.

NOT VALID

401K is not valid because of the leading digit.

100, 500, and 1000 or $100, $500, $1000 are not valid because of the leading digit and the special character '$'.

Upper camel case (UCC) MUST be used for code values. Upper Camel Case capitalizes the first character of each word and compounds the name. A leading underscore "_" sign is considered a capital character.

Example

VALID

Business Term: Misrepresentation Of Information On Application

MisrepresentationOfInformationOnApplication

NOT VALID

MisrepresentationInformationApplication

Each code value MUST be unique within the code list it belongs to, not across all code lists. Case differences do not qualify as a unique name.

Each code value in a single code list MUST represent a single semantic meaning.

Example
NOT VALID
Apple cannot appear multiple times in a single, specific code list with different meanings:
Apple = Apple Butter
Apple = Apple Tree

[R 339] There MUST not be more than one code value within a single list that represents the same thing as another code.

Example
NOT VALID
Apple = Apple Tree
Mackintosh = Apple Tree

[R 340] Once a code and definition are established, any change to the definition that changes the semantic meaning of the code MUST result in a new code.

4.4.2.3 Question Code Lists

[R 341] A question code list is a particular type of code list associated with a list of questions based upon the underwriting questions on ACORD Forms. The Question Code List Name MUST be composed of a meaningful description that identifies the question list (such as the line of business that applies to a series of underwriting forms), followed by the suffix ‘QuestionCode’.

[R 342] The code values representing a question item in a particular question code list MUST follow this format: <Question Code List Description>nnn

Example
VALID
ACORD Form 80 – Homeowners Application
Question: Any farming or other business conducted on premises? (including day/childcare)
Question code list name: HomeOwnersQuestionCode
Valid code value: HomeOwners001

[R 343] The name of a question code value (which can be specified in the instance XML document) MUST be the text of the question on the form with the following modifications.
If the question contains a statement about a number of years or months, this should be changed to remove the specific number and substitute “specified” in its place. For example, "Any losses in the past 5 years?" should be reworded to state something like "Any losses in the past specified number of years?"

If the question contains additional instructions, the instructions should not be included. These instructions might be “If yes, explain or provide date, etc.”

Example

VALID

ACORD Form 90 - Personal Auto Application

Question: Any coverage declined, cancelled, or non-renewed during last 3 years?
If yes, explain or provide date, etc

Valid display name: “Any coverage declined, cancelled, or non-renewed during specified number of years?”

4.4.3 CodeList Categories and Data Type Requirements

4.4.3.1 ACORD Closed List

An ACORD Closed List is an ACORD owned list whose values are strictly defined and are closely aligned with an ACORD Specification. A code element that is associated to this code list cannot receive values not defined in the specification version in use.

Adding a value to an ACORD Closed List requires a specification update. Since Closed Lists require close versioning in line with the overall specification, external Code Lists cannot be used for this function.

ACORD Closed Lists are assigned a specific business data type: ACORDClosedCode_Type. Since an element of this type is strictly constrained by the schema version, this data type does not define any additional attribute to specify the code name and the source or version of the code list.

4.4.3.2 ACORD Open List

An ACORD Open List is an ACORD owned list whose values are defined in an ACORD Specification. However a code element instance of this type can receive values not defined in the specification.

Values can be added to an ACORD Open List with a specification update. Custom extensions are permitted, requiring out-of-band agreement by the end points.

Example:

OccupationCode: is a code list that has a base value set defined by ACORD but that is often complemented by insurer specific values.

ACORD Open Lists are assigned a specific business data type: ACORDOpenCode_Type. Being derived from the xsd:QName schema built-in type, ACORDOpenCode_Type forces code values to be namespace qualified in support of the general ACORD XML
extensibility requirements exposed in Chapter 8. In addition, `ACORDOpenCode_Type` defines metadata attributes to specify the code name and the source of the code list. When extending an ACORD Open List the implementer should give information on the source of the non-ACORD values:

[R 344] When a code value from an external source is used to extend an ACORD Open List, the coded element SHOULD reference the `CodeList` aggregate to give details of the source and version.

[R 345] An ACORD Open Code List SHOULD NOT be extended with values from an external organization which manage a list with the same semantic meaning.

4.4.3.3 Open List

An Open List is a code list used in an ACORD Specification but whose values are maintained by an external organization. ACORD will not manage schema enumerations for non-ACORD lists but will provide complete documentation on the source of the list and the currently supported version. Implementers should contact the maintaining organizations about adding or modifying values in those lists.

Like for ACORD Open Lists, custom extensions are permitted, requiring out-of-band agreement by the end points.

Open Lists are assigned the unmodified `Code_Type` Core Data type. When using a non-ACORD Open List the implementer should give information on the source and version of the code values:

[R 346] A code element that is associated to an Open List SHOULD reference the `CodeList` aggregate to give details of the source and version.

Example:

`LossCauseCode` is a code list that can receive several value sets maintained by different external organizations (e.g. WCIO, ISO, Bureau of Labor statistics), but not by ACORD.

ACORD may import external code schema enumerations on a case-by-case basis.

[R 347] Where there are external code lists available in an acceptable (compatible) schema format and copyright and intellectual property constraints allow it, ACORD MAY import that schema for use; but the maintenance of the schema will be the responsibility of the maintaining organization.
4.4.4 Code List Specification

ACORD “centrally” manages the semantics of codes. Codes that cover the same concept must have consistent definitions, names and other properties whichever is the context where they are used.

As explained earlier a list of codes covering the same concept constitutes a “Code List”.

ACORD will initially publish its owned Code Lists in the form of schema modules containing the code value enumerations, together with annotations for code names and other properties. The detailed format of Code List schema modules is presented in Chapter 7.

ACORD has considered using another format than the schema language for code list specifications, namely the Genericode OASIS Standard [Genericode]. However this was left for possible future development.

4.4.5 Code List Validation and Implementation Support

4.4.5.1 Code Subset Lists

Implementation guidance and message instance validation require publishing the list of permissible values for each XML element in context. In the majority of cases, the Code Lists as defined above – containing the full list of codes covering the same concept – can be reused for this purpose.

When it is necessary to define restricted sets of permissible values from the same Code List depending of the context, ACORD will publish additional Code Subset List schema types derived by restriction from the main Code List schema type. The Code List and the Code Subset Lists will reside in the same schema module, as detailed in Chapter 7.

[R 348] When restricted sets of permissible values from the same Code List apply depending on the element context, ACORD MUST publish Code Subset Lists for validation and implementation guidance.

4.4.5.2 Code Validation Rules

4.4.5.2.1 ACORD Close List

As per the definition of an ACORD Closed List, a code element that is associated to this type of code list cannot receive values not defined in the specification version in use

[R 349] When processing an element based upon an ACORD Closed List, if a value other than one defined in the specification is received, it MUST be rejected as invalid.

Since an element of this type is strictly constrained by the schema version, the element must be assigned a data type that specifies the applicable code value enumeration.
[R 350] A Code element that is associated with an ACORD Closed List MUST be assigned the dedicated named type derived from the corresponding Code List or Code Subset schema type, i.e. containing the applicable code value enumeration.

4.4.5.2.2 ACORD Open Lists

As per the definition of an ACORD Open List, a code element that is associated to this type of code list can receive values not defined in the ACORD Specification version in use provided that an out-of-band agreement is made between the end points.

[R 351] When processing an element based upon an ACORD Open List, if a value other than one defined in the specification is received, it SHOULD NOT be rejected as invalid except by the final recipient or user of that code value.

ACORD is due to support validation of ACORD Open Lists taking into account the following considerations:

- A part of open lists validation may only occur only at the application level, in particular at code value translation time.
- However it is easier to write validators upfront than to write programs that behave defensively.
- Today tighter validation is required to track mistakes even for optional tags or open code lists to avoid losing information when there are spelling mistakes.
- Hardware appliances can optimize performances for standard technologies like schema or stylesheets.

Members have requested that ACORD support two options for Open List code validation:

- First pass validation with schema (including agreed code extension)
- Second pass validation by specialized validator (e.g. Schematron) or application

Arguments for second pass validation include:

- Code list volatility (updated more often than specification).
- Existence of community implementation guides, containing rules that cannot be validated by schema, including code list restrictions.

In consequence ACORD has to publish Code Lists in a way that allows automated code validation for both options. It has been decided to initially use the schema enumeration format as implemented by the Code List schema modules, which can practically be reused for second pass validation solutions.
Enforcement of schema enumerations for code elements must be optional.

[R 352] A Code element that is associated with an ACORD Open List MAY be assigned the dedicated named type derived from the corresponding Code List or Code Subset schema type, i.e. containing the applicable code value enumeration.

ACORD has considered using another format than the schema language for second pass code list validation, namely the context/value association file as defined by OASIS/UBL [UBL-Code List Methodology]. However this was left for possible future development.

ACORD has also considered enforcement of second pass validation between Trading Partners, using an explicit declaration of non-schema validation scripts in messages. This was also left for possible future development.
5 GENERAL SCHEMA LANGUAGE CONVENTIONS

5.1 General


[UNR 3] All ACORD XSD Schema and conformant XML instance documents MUST be based on the W3C suite of technical specifications holding recommendation status.

[R 353] Schema files MUST be encoded in UTF-8.

5.2 Schema Element

Locally declared elements must inherit the namespace of their schema.

[UNR 56] The @xsd:elementFormDefault attribute MUST be declared and its value set to qualified.

Locally declared attributes must not inherit the namespace of their schema.

[UNR 57] The @xsd:attributeFormDefault attribute MUST be declared and its value set to unqualified.

As explained elsewhere in the specification, and subject to future decisions on permitted schema extensibility methods the schema will be blocked for the use of substitution groups and xsi:type derivation.

[R 354] The @xsd:blockDefault attribute MUST be declared and its value set to #all.

The xsd prefix will be standardized in the schema modules.

[UNR 58] The xsd prefix MUST be used in all cases when referring to the W3C Schema language specification, as follows:

xmlns:xsd=http://www.w3.org/2001/XMLSchema

Example:

<xsd:schema targetNamespace="... see namespace ...
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified" blockDefault="#all">
5.3 Constraints on Schema Construction

ACORD doesn’t use `xsd:notation` in schemas.

[UNR 60] *xsd:notation* MUST NOT be used.

Subject to future decisions on permitted schema extensibility methods, free element extensions are not allowed.

[UNR 62] *xsd:any* MUST NOT be used.

As defined in extensibility rules (Chapter 6), attribute extensions are not allowed.

[UNR 63] *xsd:anyAttribute* MUST NOT be used.

Subject to future decisions on permitted schema extensibility methods, substitution groups are not allowed.

[UNR 65] *xsd:substitutionGroup* MUST NOT be used.

In general, the absence of an element in a document instance does not have any particular meaning - it may indicate that the information is unknown, or not applicable, or the element may be absent for some other reason.

[UNR 68] The absence of a construct or data MUST NOT carry any meaning.

5.3.1.1 Use of Recursion

XML allows an aggregate to reference itself as content; this is defined as recursion. Recursion creates an unbound set of nestings that cannot be controlled by the schema. Recursion is a natural way to represent any object that is made up of assemblies, subassemblies, and ultimately the individual parts. There is no maximum number of assemblies that might occur. Because a recursive relationship is technically unbounded, some code generation tools may fail with this design construct.

It is easy to recognize a recursive relationship where an aggregate is directly used as a child of itself. Whether the aggregate is a direct child or a descendant, both situations create a recursive arrangement.

[R 355] It is recommended that recursion SHOULD NOT be used in business data; but as this is a natural way to describe assemblies, if there is no other solution that can be found, then recursion might be used.

5.4 Attributes

5.4.1 Usage of Attributes

The use of XML attributes for business data has been deemed by ACORD membership to be inappropriate. However, it is recognized that Core Data Types and derived Business Data Types could inherently use attributes to describe metadata relating to element content (See Chapter X for examples in this specification). Attributes will also be used in a few other cases.
[R 69] - [UNR 69] XML Attributes MUST only exist on elements:

1. to convey metadata information in Core Data Types or derived Business Data Types (aka ‘supplementary’ components in the Core Component vocabulary);
2. to specify the language on an element’s content with @xml:lang; this is considered a particular case of metadata information in Core Data Types or derived Business Data Types;
3. to serve as identifiers and references when two elements need to be related to one another via schema identity constraints (aka key-keyref constraints);

However, predefined global attributes commonly used in XML may be used as described elsewhere in this document. This is the case for example:

   to declare a namespace (i.e. @xmlns);
   to declare a schema location (i.e. @xsi:schemaLocation).

5.4.2 Attribute Declaration

Best practice discourages the use of global attributes unless these attributes are reused in a variety of elements, which are in a variety of namespaces. This is because global attributes must be prefixed in instance documents. Given the prescribed usage of attributes as described above, all attributes declared in ACORD schemas will be defined locally, with the exception of the @xml:lang attribute.

[R 356] All attributes declared in ACORD schemas MUST be defined locally, with the exception of the @xml:lang attribute.

The attributes can represent different types of values. Some of the values can be variable information or can be based on code lists.

[UNR 70] A `xsd:attribute` that represents variable metadata information in Core Data Types or derived Business Data Types MUST be based on the appropriate XSD built-in data type.

[UNR 71] A `xsd:attribute` that represents coded metadata information in Core Data Types or derived Business Data Types MUST be based on the `xsd:simpleType` of the appropriate code list.

An attribute that serves as identifier or reference must typed as described elsewhere in this specification.
5.4.3 Language Specification

In general, the expected content of an ACORD document instance is expected to be in English. There may be occasions when it is needed to send a message in a different language or that specific content require a different language. Both of these situations are handled with the @xml:lang attribute.

XML is based upon the Unicode character set and as such can support a vast range of languages. To easily process the content of a document instance and detect the language being used, the @xml:lang attribute is used. The @xml:lang attribute can be used to:

- Flag a default language
- Allow for multiple languages in the same stream

The @xml:lang attribute has special properties, like a namespace declaration. The @xml:lang sets a context that can be tested at any level below its declaration. As such, when this is applied to the root of a document instance, it sets a default condition for everything in that document instance, and its value can be tested at each level; even though it has not been explicitly defined at the lower level. A local definition can override the default, and if it is declared on an aggregate, then the content of that aggregate has a new default value.

Quoting from the XML Standard [XML]: "The intent declared with xml:lang is considered to apply to all attributes, and content of the element, where it is specified, unless overridden with an instance of xml:lang on another element within that content." The @xml:lang uses the standard language codes defined in [ISO 639] and country codes from [ISO 166] to specify the country and language being used. Here are some examples:

```xml
xml:lang='en' <!-- any English -->
xml:lang='en-GB' <!-- any UK/British English -->
xml:lang='en-US' <!-- any US English -->
xml:lang='de'  <!-- any German -->
xml:lang='fr'  <!-- any French -->
```

[R 357] When defining anything related to the language used in a document instance, the @xml:lang attribute MUST be used.

[R 358] In document instances that define a default language with the @xml:lang at the top of the document instance, if this attribute is missing, "English" (a value of ‘en’) MUST be assumed for the entire document instance.

Example

```xml
<PersonalAutoPolicyNewRequest xml:lang='en'>
```
@xml:lang must be used in the definition of a Core Data Type, for the language metadata information.

[R 359] When a Core Data Type requires the specification of the language, the @xml:lang attribute MUST be used.

Language Specification should only be used for understanding a text content, i.e. when it could be translated in another language, not for fixed and untranslatable text. For example an address or a person name can have strange characters but don’t require the use of xml:lang.

[R 360] Specification of the language of a simple element text content SHOULD only be used when the language code is essential for understanding the text content.

5.5 Elements

5.5.1 Usage of Elements

In relation to UML Class diagrams, elements are used to render classes and attributes of a class. This includes message roots, aggregate and simple business data.

5.5.2 Element Declarations

5.5.2.1 Elements with Empty Content

Elements with empty content do not provide the level of semantic assurance necessary for business information exchanges and as such, will not be used. The only exception is for elements that include an identifier attribute that serves to reference another element via schema identity constraints. In the schema language empty content is simply indicated by the absence of a content model in the element declaration.

[R 74] - [UN R 74] Elements with empty content MUST NOT be declared, except when their definition include an identifier attribute that serves to reference another element via schema identity constraints; such usage MUST conform to rules described elsewhere in this document.

Note: ACORD further prevents empty content by forcing all character string based Business Data Types to have a minimum length of one.

5.5.2.1.1 Nullable Content

The XML schema specification does however provide a feature, the xsd:nillable attribute, whereby an element may be transferred with no content provided that the @xsi:nil attribute explicitly appears in the element. In accordance to the principle that no empty element should be used to retain semantic clarity, the nillability feature of XSD will not be used.
5.5.2.2 Simple Content Elements

Every simple content element must be explicitly typed with a permitted xsd built-in type, Core Data Type or Business Data Type as defined elsewhere in this specification.

5.6 Types

5.6.1 Usage Of Types

Example: Type Definition Name

```xml
<xsd:complexType name="Account_Type">
  <xsd:annotation>
    ... see annotation ...
  </xsd:annotation>
  <xsd:sequence>
    ... see element declaration ...
  </xsd:sequence>
</xsd:complexType>
```

Data types are intended to be reused to the maximum extent possible. If an existing data type has the same semantic meaning and structure (facet restrictions) as the intended data type, then the existing data type should be used rather than creating a semantically equivalent duplicate data type.

[UNR 78] Data type definitions with the same semantic meaning MUST NOT have an identical set of facet restrictions.

5.6.1.1 Simple Types

The usage of attributes is strictly controlled by this specification. As defined in extensibility rules (Chapter 6), adding custom attributes to any type is not permitted. Therefore complex types must not be defined for simple content elements when user’s business requirements do not require attributes.
xsd:simpleType MUST always be used where they satisfy the user's business requirements. Where these business requirements cannot be satisfied, complex type definitions will be used.

5.6.1.2 Complex Types

xsd:complexType MUST be used when it is not possible to satisfy the user's business requirements with a xsd:simpleType or when an aggregate business information must be defined.

5.6.2 Type Declarations

5.6.2.1 Mandatory and Optional Elements

An ACORD Specification provides cardinality information about all elements in this specification. This is optional or required along with the possibility that it might repeat.

When designing an aggregate, the cardinality of each element MUST be specified using @minOccurs and @maxOccurs attributes. Per the schema language, these attributes MAY be skipped if they take the default value of 1.

Most elements and aggregates will have to be defined as optional, because in an industry standard, there are very few data elements that are consistently used and required by all parties involved. This forces to make many things optional to handle the least restrictive of all possible uses. On top of these ACORD minimal constraints, an optional element might be part of the minimum data set that is required by a trading partner.

In some situations, this design philosophy creates a required aggregate that might have no required child elements. This is not a mistake, and the rules for sending empty content (see Chapter 6), requires that at least one child element must be sent in the document instance.

5.6.2.2 Element Grouping

All elements in a complex type must be explicitly sequenced and usage of xsd:all groups is not permitted.

The xsd:all compositor requires occurrence indicators of minOccurs = 0 or 1 and maxOccurs = 1. The xsd:all compositor allows for elements to occur in any order. The result is that in an instance document, elements can occur in any order and never occur more than once.

Such restrictions are in conflict with the requirement for schema derivation from logical models (which specifies the cardinality of elements).
In addition `xsd:complexType` elements with an `xsd:all` content cannot be extended using schema type derivation mechanisms.

[UNR 76] `xsd:all` groups MUST NOT be used when defining a complex type.

### 5.6.2.3 Mixed Content

Mixed content is described in the XML Specification [XML] as the combination of tags and text as the content of an element.

[UNR 64] An aggregate MUST not be defined with mixed content. An aggregate MUST only contain other tags, and only elements will contain data. In other words, the `@mixed` attribute MUST NOT be used in the `xsd:complexType` and `xsd:complexContent` elements.

### 5.6.2.4 Use of Schema Extension and Restriction

The concept of derivation through the use of `xsd:extension` and `xsd:restriction` will only be used in ACORD schemas in limited circumstances as described below.

#### 5.6.2.4.1 Extension

[R 79] - [UNR 79] The schema `xsd:extension` mechanism MUST only be used in ACORD schemas in the two following circumstances:

1. To mirror UML business data model inheritance associations, which indicate classes that inherit a source class; when used this MUST strictly limit to mirroring the inheritance associations as defined in the business data model.
2. To define Core Data Types, where attributes need to be added to a simple type; when used it MUST only be used for declaring attributes to accommodate relevant metadata information.

#### 5.6.2.4.2 Restriction

`xsd:restriction` will be used as appropriate to define types that are derived from existing types. Only simple types are permitted to be restricted in ACORD schemas.

[R 364] `xsd:restriction` MUST only be used for simple types to restrict facets of their content type.

Where used, the derived types must always be renamed.

[R 80] - [UNR 80] When `xsd:restriction` is applied to a `xsd:simpleType` that represents a data type the derived construct MUST use a different name.

**Example: Restriction of Simple Type**
5.6.3 Annotations

All ACORD schema constructs will use `xsd:annotation` to provide the appropriate documentation on this schema construct, as required elsewhere in this specification.

[UNR 81] Each ACORD defined or declared construct MUST use the `xsd:annotation` element for required documentation.

The format of ACORD documentation is specified in schema modules whose namespace is: http://www.acord.org/schema/documentation/standard/Annotations/1

Thus, all schema modules must contain the following namespace declaration:

```xml
xmlns:acord-doc="http://www.acord.org/schema/documentation/standard/Annotations/1"
```

and all documentation elements must be prefixed with ‘acord-doc’.

Please refer to Appendix D for a list of Annotation templates.
6 INSTANCE DOCUMENTS

6.1 ACORD Schema Compliance

In order to be ACORD conformant, an instance document must be valid against the relevant ACORD XML schema.

[Note] It is not required to perform validation while parsing a document, but it is recommended that a validating parser be used at least during design and test. ACORD uses a validating parser to certify application and sample documents.

A few additional rules are applicable in order to enhance readability and interoperability.

6.2 Character Encoding

ACORD aligns with the UN/CEFACT NDR to mandate UTF for XML instantiation. UTF-8 is the preferred encoding, but UTF-16 may be used where necessary to support other languages. Note that per the XML [XML] Specification, all XML processors are required to be able to process documents encoded using UTF-8 and UTF-16.

[UNR 198] All ACORD XML MUST be instantiated using UTF. UTF-8 should be used as the preferred encoding. If UTF-8 is not used, UTF-16 MUST be used.

6.2.1 XML Declaration

The XML declaration is an optional feature of an XML document that appears in the prolog of a document. ACORD mandates it.

[R 365] In an ACORD XML document, the XML declaration MUST be used; it MUST contain the encoding attribute that specifies the exact character encoding being used in the document.

Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

6.3 Standard Schema Reference

The W3C has defined several attributes that can be used to define/declare which schema the data stream is based upon. These attributes are defined as optional and hints in the Schema Specification, but all parsers have not treated them equally. Some parsers require these attributes to locate the proper schema; others do not. By using the schema standard attributes, the information is supplied in a form that the XML parser can use for validation if it reads these values and validation is required.

For consistent processing, the following rules will apply:

[UN R 199] The xsi prefix MUST be used where appropriate for referencing `xsd:schemaLocation` attributes in instance documents.
The following attributes MUST always be supplied:

1. xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
2. xsi:schemaLocation with the ACORD namespace and name of the ACORD schema file for the used version of the schema

6.4 Namespace Declarations

To be ACORD schema conformant, ACORD documents will only contain namespace qualified elements. Relevant namespace declarations must therefore be present.

ACORD documents MUST contain the following namespace declarations:

1. The ACORD Process Domain namespace
2. The ACORD Reusable Data Components namespace
3. Other namespaces visibly used by elements or attributes in the document instance (if any); no namespace declaration is needed for @xml:lang

All namespaces SHOULD be defined at the highest level (typically the root) element in the document.

When putting a data stream together, you should evaluate how it will be processed and if you should make use of a default namespace. The default namespace should be used for the namespace with the most use in the document. It may be best to prefix all namespaces but generally the ACORD Reusable Data Components namespace is predominant and is the recommended choice to serve as default namespace, even in the presence of custom extensions.

If a default namespace is used it SHOULD be the most used namespace in the document. This will most often be the ACORD Reusable Data Components namespace.

Example – Message root element with all the above rules applied:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<pr:LiabilityClaimNotify
    xml:lang="en"
    xmlns="http://www.acord.org/schema/data/standard/ReusableDataComponents/1"
    xmlns:pr="http://www.acord.org/schema/process/standard/PCSClaim/1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.acord.org/schema/process/standard/PCSClaim/1
    process\standard\ACORD-PCSClaimMessages-1.0.xsd">
```
6.5 Empty Content

Elements with empty content do not provide the level of semantic assurance necessary for business information exchanges and as such, will not be used. The only exception is for elements that include an identifier attribute that serves to reference another element via schema identity constraints.

[UNR 200] UN/CEFACT conformant instance documents MUST NOT contain an element devoid of content, except if this element includes an identifier attribute that serves to reference another element; such usage MUST conform to rules described elsewhere in this document.


[R 370] When an element with empty content is legally used in the ACORD document, it SHOULD use the short hand notation for an empty tag.

Example – allowed element with empty content

<Producer organizationReference='pt1'/>.

6.6 xsi:type

The xsi:type attribute allows for substitution during an instantiation of a xml document. The xsi:type attribute is not allowed.

[UNR 202] The xsi:type attribute MUST NOT be used.
7 XML SCHEMA MODULES

This section describes the detailed rules for building ACORD conformant schemas, in complement of the general rules exposed in Chapter 3.

7.1 Process Domain Schema Modules

Process Domain schemas focus on the structures that assemble Reusable Data Components to provide data content to a particular process.

The modularity model described in Chapter 3 has opted for one single schema module per Process Domain namespace.

- [R 371] Each Process Domain schema module MUST contain the definitions of all the messages that fall in the Process Domain.

- [R 83] - [UNR 83] Each Process Domain schema module MUST incorporate the following external schema modules by an xsd:import statement:
  - the ReusableDataComponents Schema Module

- [R 87] - [UNR 87] A global element known as message root element, representing the business information payload, MUST be declared for each message that fall in the Process Domain in the Process Domain schema.

- [R 90] - [UNR 90] The Process Domain schema module MUST define a single xsd:complexType that fully describes each message business information payload.

- [R 89] - [UNR 89] A message root element declaration must be of xsd:complexType that represents the business information payload.

- [R 91] - [UNR 91] The name of the message root element xsd:complexType MUST be the name of the root element with the word '_Type' appended.

Please refer to Appendix C for a Process Domain schema module template.
7.2 Reusable Data Component Schema Modules

Reusable Data Component schema modules focus on the definitions of Reusable Data Components independently of their assembly in a message. There is one single Reusable Data Components namespace.

The modularity model described in Chapter 3 has opted for splitting the Reusable Data Components namespace in several internal schema modules as follows:

The Business Data Aggregates module: contains the aggregate definitions of the business data objects used in Insurance, as generally defined outside the context of a message.

The Common Message Aggregates module: contains the aggregate definitions of elements reused in the common structure of ACORD messages, e.g. those needed to add transactional information to the message, message grouping structures, or references to binary attachments.

The Business Data Type module: will contain all the ACORD reusable simple content data types.

7.2.1 Reusable Data Component Schema Set

[R 95] - [UNR 95] The Reusable Data Components namespace MUST be represented in all ACORD schemas by the token ac, used as a namespace prefix.

[R 372] A root schema module for the Reusable Data Components namespace MUST be created; it MUST be called ReusableDataComponents

[R 373] An internal schema module for the Business Data Aggregates MUST be created; it MUST be called BusinessDataAggregates

[R 374] An internal schema module for the Common Message Aggregates MUST be created; it MUST be called CommonMessageAggregates

[R 375] An internal schema module for the Business Data Types MUST be created; it MUST be called BusinessDataTypes

[R 376] The ReusableDataComponents root schema module MUST incorporate the following internal schema modules by a xsd:include statement:
7.2.2 Business Data Aggregates Schema Module

The Business Data Aggregates internal schema module will contain the aggregate definitions of the business data objects used in Insurance, as derived from the UML or Core Component syntax-neutral models they reflect.

7.2.2.1 General Rules

[R 96] - [UNR 96] The BusinessDataAggregates schema module MUST incorporate the following internal schema modules by a xsd:include statement:
- the BusinessDataTypes Schema Module

[R 102] - [UNR 102] For every object class identified in the syntax-neutral model, a xsd:element MUST be globally declared.

Note: The above rule is set up to simplify generation of schemas. The hybrid schema model prescribes that some complex elements will be declared locally in the case of a composition association. So even if all complex elements will have a global declaration in the schema, not all will be reused in the construction of other complex types.

[R 103] - [UNR 103] The name of the globally declared aggregate xsd:element MUST be derived from the name of the object class it represents, as exposed in the general naming rules specified in Chapter 4.

[R 97] - [UNR 97] For every object class identified in the syntax-neutral model, a named aggregate xsd:complexType MUST be defined.

[R 98] - [UNR 98] The name of the aggregate xsd:complexType MUST be equal to the name of the xsd:element representing the object class, with the '_Type' suffix.
[R 99] - UNR 99] Every aggregate `xsd:complexType` definition content model MUST use the `xsd:sequence` and/or `xsd:choice` elements to reflect each property of its class.

[R 100] - [UNR 100] Recursion of `xsd:sequence` and/or `xsd:choice` SHOULD NOT occur.

[R 377] Named groups MAY be created if desired, for reusability. Names for groups SHOULD be based upon the description of what they are grouping and why; followed by the following suffix:

1. sequence - `_Sequence`
2. choice or exclusive OR - `_Choice`
3. inclusive OR - `_Or` (combination of sequence and choice)

[R 101] - [UNR 101] The order and cardinality of the elements within an aggregate `xsd:complexType` MUST be according to the structure of the object class as defined in the model.

[R 104] - [UNR 104] Every globally declared aggregate `xsd:element` MUST be of the `xsd:complexType` that represents the object class.

[R 105] - [UNR 105] For every simple property of an object class, a `xsd:element` with simple content MUST be locally declared within the `xsd:complexType` representing that object class.

[R 106] - [UNR 106] The name of the `xsd:element` representing the simple property must be constructed according to the general naming rules exposed in Chapter 4.

For every associated object class whose association with parent object class is a composition, an `xsd:element` MUST be locally declared within the `xsd:complexType` representing that object class.

The name of the locally declared aggregate `xsd:element` MUST be equal to the name of the globally declared `xsd:element` representing the same object class.

Every locally declared aggregate `xsd:element`, MUST be of the `xsd:complexType` that represents its object class.

For every aggregate `xsd:element` whose association with parent is NOT a composition, the globally declared element for the associated object class is included in the content model of the parent as a nested complex property.

For every associated object class whose association with parent object class is NOT a composition, the globally declared element for the associated object class MUST be referenced using `xsd:ref`.

7.2.2.2 XML Referencing

7.2.2.2.1 Introduction

As exposed in Section 4.1.4 XML referencing mechanisms may be used to enforce the relationship between two complex elements in an XML document.

Except the Core Data Types where referencing mechanisms can also be used, this specification approaches the normalization of XML referencing constructs using the concept of ‘Object Role’. Objects Roles are used in data modeling to relate independent objects to one another. Let us take the following example: Person ‘John Smith’ is a witness in the context of Claim ‘Vehicle Accident #101”. In this example the object class Person is said to play the role of Witness in the object class Claim. Since this relationship must be implemented in UML only via compositions or aggregations, Witness will be defined as its own class: the ‘Role Class’. This will result in the following XML instances depending on whether the Person properties are nested or referenced in the Claim object/aggregate via the Witness object/aggregate:

**Nesting**

```
<Claim>
  <Witness>
    <Person>
      <Name>John Smith</Name>
    <Person>
```

© 2013
In many cases, Object Roles require specific properties, which need to be modeled in the Role Class itself. In the example this can be done by extending the Witness class with these properties, which translates to extending the Witness schema complex type similarly.

### 7.2.2.2.2 Role Classes and Schema Generation Process

In order to allow the XML schema generation process to select between the nesting and referencing methods for the implementation of a Role Class, this specification assumes that a list of predefined reference-able global elements exists in the syntax-neutral model.

### 7.2.2.3 Design and Naming Conventions for XML Referencing Constructs

This section describes the normalized XML referencing constructs that must be used to link a reference-able global element to a Role Class.

The following principles are taken into account for the implementation of key-keyref constraints:

1. Identifiers used for referencing should be attributes, to stay separate from the element’s content model.
2. For maximal element and type reuse and to stay away from forward compatibility problems, attributes used as identifiers or references should be optional. No xsd:key constraints should be defined on identifiers, which would make the identifiers mandatory in the context of a message; only xsd:unique constraints should be used.
[R 310] An optional identifier attribute named ‘@key’ MUST be defined for each aggregate. @key MUST be a reserved name. The base type of this attribute MUST be a non-null normalized string limited to 255 characters, to avoid unnecessary format constraints while limiting the length.

**Example:**

The identifier attribute of Person is Person/@key.

[R 311] An attribute used as a reference MUST have the same name as the element to which it refers, with the suffix ‘Reference’.

**Example:**

The reference to Person in the role of Witness will be Witness/@PersonReference.

The base Role Class construct must take the form of an empty element with an attribute that has the name of the referenced element, with the “Reference” suffix.

[R 379] For every predefined reference-able global element an xsd:complexType MUST be declared to define the base structure of the set of the applicable Role Classes. The structure will be that of an empty element with an attribute. The name of the xsd:complexType MUST be composed of the name of the referenced element followed by the suffix ‘Role_Type’ and the name of the attribute MUST be composed of the referenced element, followed by the suffix ‘Reference’. This xsd:complexType MUST be defined as ‘abstract’ and cannot be directly reused in the definition of elements.

**Example: ‘Person Role’ abstract type**

```xml
<xsd:complexType name="PersonRole_Type">
  <xsd:complexContent>
    <xsd:extension base="ACORDAggregate_Type">
      <xsd:attribute name="personReference" type="SimpleIdentifier_Type"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```
A variation of the rule above occurs when the referred object is one of the subtypes of a more general object. In that case, it will be possible to specify more than one referencing attributes, each of them pointing to a subtype.

[R 379a] A particular case occurs when several predefined reference-able global elements represents objects that are subtypes of a more generic object and the applicable Role Classes need to be able to reference a selection of these mutually exclusive objects. In that case the `<xsd:complexType>` that defines the base structure of the Role Classes will be constructed as follows: (1) the name of the `<xsd:complexType>` MUST be composed of the name of the more generic object followed by the suffix ‘Role_Type’ and (2) its structure will be that of an empty element with several attributes and the name of the attributes MUST be composed of the referenced elements, followed by the suffix ‘Reference’. This `<xsd:complexType>` MUST be defined as ‘abstract’ and cannot be directly reused in the definition of elements.

Example: ‘Party Role’ abstract type

```xml
<xsd:complexType name="PartyRole_Type">
  <xsd:complexContent>
    <xsd:extension base="ACORDAggregate_Type">
      <xsd:attribute name="personReference" type="SimpleIdentifier_Type"/>
      <xsd:attribute name="organizationReference" type="SimpleIdentifier_Type"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Elements used to represent the Role Object must be named according the role. There are two subcases:

[R 380] The element used to represent the Role Object MUST have a name equal to the name of the ‘Role’ played by the referenced element.

Example:

Claim has **Witness**, which is a named Role of **Person**
The referencing element will be **Witness**

[R 381] When there is no specific role name, the name of the element used to represent the Role Object MUST be composed as follows: `Referred+<Referenced Element Name>`
Example:

Hangar has CommercialLocation without identification of the Role
The referencing element will be ReferredCommercialLocation

**Note:** Although this is beyond the scope of this specification and should be a feature of the source syntax-neutral data model, it is very strongly RECOMMENDED for semantic clarity to describe the role of the referred element with a meaningful name and avoid generically named references whenever possible.

[R 382] An element used to represent a Role Object MUST be defined by a specific `xsd:complexType` whose name is equal to the name of the `xsd:element`, with the ‘_Type’ suffix. This `xsd:complexType` MUST be derived by extension from the base Role Class `xsd:complexType` corresponding to the referenced element.

Example – `xsd:complexType` derivation without additional properties

```xml
<xsd:complexType name="Witness_Type">
  <xsd:complexContent>
    <xsd:extension base="PersonRole_Type"/>
  </xsd:complexContent>
</xsd:complexType>
```

Example – `xsd:complexType` derivation with additional properties

```xml
<xsd:complexType name="Contact_Type">
  <xsd:complexContent>
    <xsd:extension base="PersonRole_Type">
      <xsd:sequence>
        <xsd:element name="PrimaryIndicator" type="xsd:boolean"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

7.2.2.2.4 Rules for Implementation of Identity Constraints

Pairs of identifier and reference attributes need declaration of identity constraints in the schema. The naming conventions exposed below are designed to simplify the declaration of the identity constraints. The rules are summarized as follows: for each referenced element in the scope of a scoping element, one `xsd:unique` and one
**xsd:keyref** constraint, with a fixed pattern, must be declared in the schema, under this scoping element (Scoping element means an element in the hierarchy of the XML document under which a closed set of references can be defined; this element can be the message root element or a descendant – see illustration of this in the example below).

[R 383] For each referenced element in a given scope one **xsd:unique** constraint involving the identifier attribute of the referenced element MUST be declared in the schema, under the scoping element. The name of the **xsd:unique** constraint MUST conventionally be composed as follows: "<scoping element name><referenced element name>Key" so that the name is unique in the schema. This declaration will guarantee uniqueness of the identifier attribute values across all referenced elements of the same name, in the given scope.

**Example:**

The following declaration under the message root element will guarantee uniqueness of the @key attribute values across all ac:Person elements, in the scope of the pr:LiabilityClaimNotify message:

```xml
<xsd:unique name="LiabilityClaimNotifyPersonKey">
  <xsd:selector xpath="ac:Person"/>
  <xsd:field xpath="@key"/>
</xsd:unique>
```

**Note:** The value of **xsd:selector/@xpath** identifies instances of one element in one namespace (by default the ACORD data namespace). Referenced elements defined in a custom extension namespace need to wear the proper namespace prefix.

[R 384] For each referenced element in a given scope one **xsd:keyref** constraint involving the reference attribute that point to this referenced element MUST be declared in the schema, under the scoping element. Since the schema will specify which parent element can contain the reference attribute, there needs only be one **xsd:keyref** constraint declared for all the elements where the reference attribute may appear. The name of the **xsd:keyref** constraint MUST be composed as follows: <scoping element name>< reference attribute name>.

**Example:**
The following declaration under the message root element will enforce referencing between all the elements that have the @PersonReference attribute and instances of Person, in the scope of the LiabilityClaimNotify message.

```xml
<xsd:keyref name="LiabilityClaimNotifyPersonReference" refer="LiabilityClaimNotifyPersonKey">
  <xsd:selector xpath="./**"/>
  <xsd:field xpath="@PersonReference"/>
</xsd:keyref>
```

**Note:** The value of `xsd:selector/@xpath` allows for any element in any namespace, including custom extensions, to be the parent element of the reference attribute in the `xsd:keyref` constraint.

Dynamic referencing does not require the schema to enforce uniqueness of @key attributes when they are not involved in structural referencing. This will avoid unnecessary complexity of the identity constraints.

[R 385] Uniqueness of @key attributes that are not involved in structural referencing MUST NOT be enforced by the schema via identity constraints. Uniqueness of @key attributes should be assured by use of adequate algorithms for the generation of the identifiers (e.g. UUIDs).

### 7.2.2.2.5 Full Example

This fictitious message example shows identity constraints defined at two levels:

- **ac:CodeList, ac:Organization and ac:Person** are referenced elements in the scope of the whole message
- **ac:AdditionalInterest and ac:Beneficiary** are referenced elements in the restricted scope of the ac:LiabilityClaim element.

#### 7.2.2.2.5.1 XML Message

```xml
<LiabilityClaimNotify xmlns="http://www.acord.org/schema/process/draft/PCSClaim/1"
  xmlns:ac="http://www.acord.org/schema/data/draft/ReusableDataComponents/1">
  <ac:CodeList key="Code1">
    <ac:ListName>-</ac:ListName>
  </ac:CodeList>
  <ac:Organization key="org1">
    <ac:OrganizationName>
      </ac:OrganizationName>
  </ac:Organization>
</LiabilityClaimNotify>
```
<ac:FullName>INSURANCE COMPANY</ac:FullName>

</ac:OrganizationName>

<ac:FEINId>785902378</ac:FEINId>

</ac:Organization>

<ac:Person key="person1">

    <ac:FullName>JOHN SMITH</ac:FullName>

</ac:Person>

<ac:Person key="person2">

    <ac:FullName>JANE JONES</ac:FullName>

</ac:Person>

<ac:ClaimPartialPolicy>

    <ac:Insured key="insured1" personReference="pers1">

        <ac:InsuredIdentifiers>

            <ac:AgencyId>-</ac:AgencyId>

        </ac:InsuredIdentifiers>

    </ac:Insured>

</ac:ClaimPartialPolicy>

<ac:LiabilityClaim>

    <ac:AdditionalInterest key="ClaimParty1" personReference="person1">

        <ac:AdditionalInterestPayorIndicator>true</ac:AdditionalInterestPayorIndicator>

    </ac:AdditionalInterest>

    <ac:Beneficiary key="ClaimParty2" organizationReference="pers1">

        <ac:BeneficiaryDependencyCode>BeneficiaryDependencyCode</ac:BeneficiaryDependencyCode>

    </ac:Beneficiary>

</ac:LiabilityClaim>
7.2.2.2.5.2 Identity Constraints Defined Under Message Root Element

These constraints enforce that references to `<ac:CodeList>`, `<ac:Organization>` and `<ac:Person>` appearing under the instance of the message root element can only refer to instances of `<ac:CodeList>`, `<ac:Organization>` and `<ac:Person>` appearing under the same message root element.

```
<xsd:element name="LiabilityClaimNotify" type="LiabilityClaimNotify_Type">
    <xsd:unique name="LiabilityClaimNotifyCodeListKey">
        <xsd:selector xpath="ac:CodeList"/>
        <xsd:field xpath="@key"/>
    </xsd:unique>

    <xsd:unique name="LiabilityClaimNotifyPersonKey">
        <xsd:selector xpath="ac:Person"/>
        <xsd:field xpath="@key"/>
    </xsd:unique>

    <xsd:keyref name="LiabilityClaimNotifyCodeListReference" refer="LiabilityClaimNotifyCodeListKey">
        <xsd:selector xpath=".//**"/>
        <xsd:field xpath="@codeListReference"/>
    </xsd:keyref>

    <xsd:unique name="LiabilityClaimNotifyPersonKey">
        <xsd:selector xpath="ac:Person"/>
    </xsd:unique>
</xsd:element>
```
7.2.2.2.5.3 Identity Constraints Defined Under <ac:LiabilityClaim> Element

These constraints enforce that references to <ac:AdditionalInterest> and <ac:Beneficiary> appearing under one instance of the <ac:LiabilityClaim> element can only refer to instances of <ac:AdditionalInterest> and <ac:Beneficiary> appearing under the same <ac:LiabilityClaim> element.

```xml
<xsd:element name="LiabilityClaim" type="LiabilityClaim_Type">
  <xsd:unique name="LiabilityClaimAdditionalInterestKey">
    <xsd:selector xpath="ac:AdditionalInterest"/>
    <xsd:field xpath="@key"/>
  </xsd:unique>

  <xsd:keyref name="LiabilityClaimAdditionalInterestReference" refer="LiabilityClaimAdditionalInterestKey">
    <xsd:selector xpath=".//"/>
    <xsd:field xpath="@additionalInterestReference"/>
  </xsd:keyref>
</xsd:element>
```
7.2.2.2.6 Automated Generation of Identity Constraints

Identity constraints must be declared in the schema in every scoping element (defined as an element in the hierarchy of the XML document under which a closed set of references can be defined). For consistency and ease of maintenance, these constraints need to be defined systematically or generated automatically. It is desirable to add documentation in the ACORD schemas so that this automatic generation can be done even if the schema is customized.

The identity constraints as ruled above can be routinely defined if, for each scoping element, there is a documented list of referenced child elements. It is proposed that this list is implemented in the complex type definition of the scoping element, using `xsd:annotation/xsd:appinfo`. For each descendant `xsd:element` of a `xsd:complexType`, which can be referenced by another descendant element of the same `xsd:complexType`, the following `xsd:annotation/xsd:appinfo` insertion will indicate that the element can be referenced and therefore may trigger an identity constraint for a scoping element that is defined by this complex type:

```xml
<xsd:annotation>
    ...
    <xsd:appinfo>
        <acord-doc:KeySelector>xpath selector</acord-doc:KeySelector>
    ...
    repeating
</xsd:appinfo>
</xsd:annotation>
```
Example:

Here is the definition of the `ac:LiabilityClaim_Type` complex type that will trigger the identity constraints for the `ac:LiabilityClaim` element that was used as example above:

```xml
<xsd:complexType name="LiabilityClaim_Type">
   <xsd:annotation>
      <xsd:appinfo>
         <acord-doc:KeySelector>ac:AdditionalInterest</acord-doc:KeySelector>
         <acord-doc:KeySelector>ac:Beneficiary</acord-doc:KeySelector>
      </xsd:appinfo>
   </xsd:annotation>
</xsd:complexType>
```

Note: This solution, which has been proved to work, is only proposed as a starting point in the context of this Candidate Recommendation. More optimal solutions may possibly be discovered during the pilot phase.

7.2.3 Common Message Aggregates Schema Module

All rules defined for the Business Aggregates schema module apply here, except [R 96], which is replaced by:

>[R 399] The `CommonMessageAggregates` schema module MUST incorporate the following internal schema modules by a `xsd:include` statement:

- the BusinessDataTypes Schema Module

7.2.4 Business Data Types Schema Module

The `BusinessDataTypes` internal schema module provides the compatibility layer with the UN/CEFACT cross-industry Core Data Type library for simple content data types.

This schema contains all the ACORD reusable simple content data types. All ACORD simple content elements MUST use a data type from this list. They are collectively called Business Data Types.

Every Business Data Type must be derived by restriction from another Business Data Type or directly from an ACORD Core Data Type or from one of the `xsd` data types authorized as an ACORD Core Data Type.
By extension, the BusinessDataTypes schema module will additionally contain:

- The root abstract data type from which all ACORD aggregate xsd:complexType definitions will be derived by extension.
- One or several named attribute groups reused in the definition of Business Data Types

The Business Data Types schema module is composed of several sections in the following order:

7.2.4.1 Core Data Type Definitions
This section lists the ACORD Core Data Types. They constitute a selection of the cross-industry Core Data Types as defined by UN/CEFACT, as defined elsewhere in this specification.

Core Data Types are named simple types or complex types with simple content. Where built-in schema data types were equivalent to named UN/CEFACT Core Data Types, they have been preferred and authorized as ACORD Core Data Types.

Some of the UN/CEFACT Core Data Types cannot directly be reused in ACORD but must be restricted to be usable. These Core Data Types are marked as abstract types.

7.2.4.2 Restricted Core Data Type Definitions Excepting Code Types
This section lists the data type derived by restriction from another ACORD business data type or directly from an ACORD Core Data Type or from one of the xsd data types authorized in the ACORD Core Data Type list, excluding the data types used for code elements.

[R 386] Where required to restrict to the content model of an existing Business Data Type, Core Data Type or one of the xsd data types authorized as an ACORD Core Data Type, a new named Business Data Type MUST be defined in the BusinessDataTypes schema module.

[R 387] The name of a Business Data Type MUST be the name of its base Business or Core Data Type with a qualifier term added. When the Business Data Type is derived from a xsd data type, its name MUST be based on the element name suffix normalized for that xsd data type.
7.2.4.3 Identifier Data Type Definitions
This section lists the identifier data types whose structure requires tight schema validation (e.g. via a pattern facet constraint).

7.2.4.4 Code Data Type Definitions
This section lists the specific Business Data Types required for code elements.

[R 388] The BusinessDataTypes schema module MUST incorporate the following schema module by a xsd:include statement:
- The ACORD_CodeLists Schema Module

[R 388a] The BusinessDataTypes schema module MAY incorporate external Code Content Type schema modules by a xsd:import statement with the appropriate namespace.

[R 389] Every code element MUST have a correspondent code data type defined, so that it is possible to restrict its content by an enumeration of values. The name of the data type MUST be: <code list name>_Type

[R 390] When a restricted set of permissible values from the same Code List apply depending on the element context a distinct data type must be defined for the correspondent Code Subset List. The name of the data type MUST be: <code subset list name>_Type

Note: For the definition of Code Subset Lists, see Section 4.4.5.

[R 390a] Every code business data type MUST be defined as a derivation of the corresponding Code List or Subset List xsd:simpleType. This derivation MUST extend the xsd:simpleType with the attributes specified for code data types.

Example - code business data type definition based on an ACORD defined enumeration

```
<xsd:complexType name="MaritalStatusCode_Type">
  <xsd:simpleContent>
    <xsd:extension base="MaritalStatusCodeContent_Type">
      <xsd:attributeGroup ref="CodeAttribute_Sequence"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```
Example - code business data type definition based on an externally defined enumeration

```xml
<xsd:complexType name="CurrencyCode_Type">
  <xsd:simpleContent>
    <xsd:extension base="clm54217:CurrencyCodeContentType">
      <xsd:attributeGroup ref="CodeAttribute_Sequence"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

7.2.4.5 Other

By extension, the BusinessDataTypes schema module will additionally contain:

- The root abstract data type from which all ACORD aggregate `xsd:complexType` definitions will be derived by extension.
- One or several named attribute groups reused in the definition of Business Data Types

7.2.5 Code List Schema Module

The Code List schema module is composed of all the ACORD defined Code Enumeration simple types.

The Code List Specification and validation model described in Section 4.4 has opted for creating one Code Enumeration simple type per Code List or Code Subset List, i.e. per specific code value enumeration that can be applied to an ACORD code element.

For both Code List Specification and validation purpose, ACORD will create two variants of the Code List schema module: the open code lists module containing all code enumerations (named `ACORD_CodeLists`) and the closed code lists module containing the closed code enumerations only (named `ACORD_CodeLists_ClosedEnum`).
7.2.5.1 Content of Code List Schema Module

[UNR 171] The Code List schema modules MUST not import or include any other schema modules.

[R 172] - [UNR 172] Within the Code List schema module each Code List and Code Subset List MUST define one, and only one, named \texttt{xsd:simpleType} to specify the value facets and enumerations of the list.

\textbf{Note:} For the definition of Code Subset Lists, see Section 4.4.5.

[R 173] - [UNR 173] The name of the \texttt{xsd:simpleType} MUST be the name of the Code List or Code Subset List with the suffix ‘\texttt{Content\_Type}’ appended.

[R 174] - [UNR 174] The \texttt{xsd:restriction} element base attribute value MUST be set to \texttt{xsd:QName}.

[R 175] - [UNR 175] Each code in the Code List or Subset List MUST be expressed as an \texttt{xsd:enumeration}, where the \texttt{xsd:value} for the enumeration is the actual code value.
8 EXTENSIBILITY AND CUSTOMIZATION

All ACORD Specifications must be customizable. To make this customization interoperable between trading partners, ACORD has created a set of rules for what the document instance must look like after customization. ACORD does not specify at this time how to use the schema language or other standards to design, publish and manage customized specifications. The primary goal is that the resulting document instance can be processed by anyone without regard of the method used to design, publish and manage customizations.

It is however agreed among ACORD members that the next goal should be the standardization of how to use the schema language or other standards to design, publish and manage customized specifications. A set of initial requirements can be found in Appendix F.

8.1 Customization Rules for Document Instance

[R 393] All custom new elements MUST be identified with a namespace other than the ACORD namespace.

**Note:** With hybrid schemas as ruled by this specification, in order to identify customized data by a namespace, one must fully change the namespace of a global element and its children if only one of the locally defined children elements has been changed.

[R 394] Custom aggregates SHOULD reuse existing elements, code lists, and data types from the ACORD Specification. They SHOULD be the result of extending the ACORD simple and complex types rather than creating completely new custom structures.

[R 395] All new code values added to an ACORD list MUST be identified with a namespace other than the ACORD namespace. Code values defined in a new custom code list do not require the use of a prefix, as they will inherit the namespace of the custom code list.

[R 396] When creating new objects, you SHOULD follow the ACORD naming and design rules. This adds consistency to the design and makes it easy for all recipients to work with your customizations.
8.1.1 Customization That is NOT Allowed

The following features are not allowed in an ACORD compliant document instance:

[R 397] Any customization used MUST NOT introduce schema instance attributes beyond those already used by ACORD. For instance @xsi:type, any technique that required this to be used in the data stream is not allowed.

[R 398] You MUST NOT create new attributes unless these are used for referencing a custom new element. You SHOULD follow the ACORD naming and design rules to create these attributes.
9 REFERENCES


http://docs.oasis-open.org/ubl/prd-UBL-NDR-2.0.pdf

http://www.openapplications.org/downloads/oagis/loadfrm90NDR.htm

http://docs.oasis-open.org/codelist/cd-genericode-1.0/doc/oasis-code-list-representation-genericode.pdf

[UBL-Code List Methodology] UBL Methodology for Code List and Value Validation Working Draft 0.8, OASIS, 23 April 2007  

[ISO 11179-5] Standardization of data elements - Naming and identification principles

[ISO 166] Country Codes

[ISO 4217] ISO 4217:2001 Codes for the representation of currencies and funds

[ISO 639] Language Codes


[XPath2] XPath 2

[XSLT2] XSLT—2
**10 GLOSSARY**

**Aggregate** An Aggregate is a collection of elements and/or other aggregates. An aggregate may not contain any data itself, but rather contains elements containing data, and/or recursively contains aggregates.

This is an ACORD term.

**Aggregation** An Aggregation is a special form of Association that specifies a whole-part relationship between the aggregate (whole) and a component part.

**Attribute (XML)** An XML Attribute is considered to be meta-data about the content of the element that it is associated with. In this document the use of attributes is limited to creating links or relationships between objects in the data stream.

**Business Semantic(s)** A precise meaning of words from a business perspective.

**Cardinality** An indication whether a characteristic is optional, mandatory and/or repetitive.

**Complex data type** A complex data type is contrasted with the simple data types. Where a simple data type has a single value and defined type; a complex data type requires more than one piece of information to complete the description.

**Composition** A form of aggregation which requires that a part instance be included in at most one composite at a time, and that the composite object is responsible for the creation and destruction of the parts. Composition may be recursive.

**Controlled Vocabulary** A supplemental vocabulary used to uniquely define potentially ambiguous words or Business Terms. This ensures that every word within any of the Core Component names and definitions is used consistently, unambiguously and accurately.

**Core Component (CC)** A building block for the creation of a semantically correct and meaningful information exchange package. It contains only the information pieces necessary to describe a specific concept.

**Data stream / Instance document** A data stream or instance document is a collection of elements and/or aggregates that constitutes a well-formed XML document.

**Element** An Element is the most basic unit of data in the ACORD Specification. An element is defined based on one of the supported data types to define a single piece of information passed between the client and server. An element is named according to specific rules and has a definition associated with it to provide additional information on what it contains. An element may also have some usage rules associated with it, which describe how the client and server...
interact with the element.

An ACORD document contains one or more elements. An element includes a data value bound by any start/end tags. So, when an element is stated as required, the data value must be included (i.e. Not NULL).

An ACORD document contains one or more elements. An element is some data bounded by a leading start tag and a trailing end tag. For example, an element "Tag," containing data "SomeData," would look like this:

```xml
<Tag>SomeData</Tag>
```

Note that this definition differs slightly from the World Wide Web Consortium (W3C) XML definition of element in that an ACORD element must contain data, but may not contain other elements. A W3C XML element containing other elements is defined in ACORD as an aggregate. The W3C is the worldwide standards body for web technology.

An element contains only data. This is the leaf or bottom of the tree.

Elements contain simple data of the types shown in §5. Data Types.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower-Camel-Case (LCC)</td>
<td>A style that capitalizes the first character of each word except the first word and compounds the name.</td>
</tr>
<tr>
<td>Namespace</td>
<td>A collection of names, or XML vocabulary, identified by a URI reference.</td>
</tr>
<tr>
<td>Namespace name</td>
<td>The URI value of the @xmlns attribute.</td>
</tr>
<tr>
<td>Namespace prefix</td>
<td>A value or name assigned to a namespace for identification; a shorthand reference for the URI.</td>
</tr>
<tr>
<td>Object Class</td>
<td>The logical data grouping (in a logical data model) to which a data element belongs (ISO11179).</td>
</tr>
<tr>
<td>Object Class Term (Core Component)</td>
<td>A component of the name of a Core Component which represents the Object Class to which it belongs.</td>
</tr>
<tr>
<td>Property Term (Core Component)</td>
<td>A semantically meaningful name for the characteristic of the Object Class that is represented by the Core Component Property.</td>
</tr>
<tr>
<td>Qualifier Term (Core Component)</td>
<td>A word or group of words that help define and differentiate an item (e.g. a Business Information Entity or a Data Type) from its associated items (e.g. from a Core Component, a Core Component Type, another Business</td>
</tr>
</tbody>
</table>
Information Entity or another Data Type).

**Representation Term (Core Component)**
The type of valid values for a Basic Core Component or Business Information Entity.

**Schema module**
An XML file that contains a set of schema components that constrain the contents of instances. A schema module is an XML document whose root element is `<schema>` in the XML Schema Definition Language namespace. The term "schema module" is equivalent to "schema document" as used in the W3C Specification.

**Schema set**
The set of schema modules that altogether form the XML vocabulary identified by a namespace. This term is equivalent to the generic term “schema” as used in the W3C Specification.

**Simple data type**
A simple or atomic type with a specific data type like: decimal, integer or boolean.

**Tag**
A Tag is the generic name for either a start tag or an end tag. An element is delimited with start and end-tags. A Start Tag consists of a field name surrounded by angle brackets. An End Tag is the same as a start tag, with the addition of a forward slash immediately preceding the field name. For example, the start tag for the field named "Tag" looks like this:

```
<Tag>
```

while the end tag for the same field looks like this

```
<Tag/>
```

**Upper-Camel-Case (UCC)**
A style that capitalizes the first character of each word and compounds the name.

**URI**
Uniform Resource Identifier, possible examples are a web page address, e-mail address, FTP site, etc.

**W3C**
World Wide Web Consortium, the organization under which XML was developed and maintained (among other standards for the Internet).
## 11 Revision History/Change Summary

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.0 Candidate Recommendation - for approval</td>
<td>November 2007</td>
<td>Candidate Recommendation – for approval version.</td>
</tr>
<tr>
<td>1.0.1 Candidate Recommendation</td>
<td>January 2008</td>
<td>Candidate Recommendation release. The text of the specification for approval has been reread and reformatted using technical writing standards. No change in content.</td>
</tr>
<tr>
<td>No change</td>
<td>February 2009</td>
<td>*ACORD's Standards License (formerly our Terms and Conditions of Use) has been updated. This is a documentation change only. No content changes have been made.</td>
</tr>
</tbody>
</table>
# A. ACORD APPROVED ACRONYMS

## List of Allowed Acronyms and Their Meaning

<table>
<thead>
<tr>
<th>Expended term</th>
<th>Source</th>
<th>Acronym</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Personal Injury Protection</td>
<td>USA</td>
<td>APIP</td>
</tr>
<tr>
<td>All Terrain Vehicle</td>
<td>USA</td>
<td>ATV</td>
</tr>
<tr>
<td>American Association of Insurance Services</td>
<td>USA</td>
<td>AAIS</td>
</tr>
<tr>
<td>American Association of Motor Vehicle Administrators</td>
<td>USA</td>
<td>AAMV/Anet</td>
</tr>
<tr>
<td>American Stock Exchange</td>
<td>USA</td>
<td>AMEX</td>
</tr>
<tr>
<td>Amplitude Modulation</td>
<td>International</td>
<td>AM</td>
</tr>
<tr>
<td>Association Canadienne de Normalisation</td>
<td>International</td>
<td>ACNOR</td>
</tr>
<tr>
<td>Association for Cooperative Operations Research and Development</td>
<td>International</td>
<td>ACORD</td>
</tr>
<tr>
<td>Cable Television</td>
<td>International</td>
<td>CATV</td>
</tr>
<tr>
<td>California Workers Compensation</td>
<td>USA</td>
<td>CAWC</td>
</tr>
<tr>
<td>Canadian Life Electronic Data Interchange Standards</td>
<td>International</td>
<td>CLIEDIS</td>
</tr>
<tr>
<td>Canadian Standards Association</td>
<td>International</td>
<td>CSA</td>
</tr>
<tr>
<td>Carbon Dioxide</td>
<td>International</td>
<td>CO2</td>
</tr>
<tr>
<td>Certified Public Accountant</td>
<td>International</td>
<td>CPA</td>
</tr>
<tr>
<td>Chief Executive Officer</td>
<td>International</td>
<td>CEO</td>
</tr>
<tr>
<td>Chief Financial Officer</td>
<td>International</td>
<td>CFO</td>
</tr>
<tr>
<td>Chief Operating Officer</td>
<td>International</td>
<td>COO</td>
</tr>
<tr>
<td>Closed Circuit Television</td>
<td>USA</td>
<td>CCTV</td>
</tr>
<tr>
<td>Collision Industry Electronic Commerce Association</td>
<td>International</td>
<td>CIECA</td>
</tr>
<tr>
<td>Commercial Lines Manual</td>
<td>USA</td>
<td>CLM</td>
</tr>
<tr>
<td>Compensation Advisory Organization of Michigan</td>
<td>USA</td>
<td>CAOM</td>
</tr>
<tr>
<td>Customer Profile Exchange</td>
<td>International</td>
<td>CPX</td>
</tr>
<tr>
<td>Delaware Compensation Rating Bureau</td>
<td>USA</td>
<td>DCRB</td>
</tr>
<tr>
<td>Department Of Motor Vehicles</td>
<td>USA</td>
<td>DMV</td>
</tr>
<tr>
<td>Department Of Transportation</td>
<td>USA</td>
<td>DOT</td>
</tr>
<tr>
<td>Depository Trust and Clearing Corporation</td>
<td>USA</td>
<td>DTCC</td>
</tr>
<tr>
<td>Drive Other Car</td>
<td>USA</td>
<td>DOC</td>
</tr>
<tr>
<td>Dwelling Policy</td>
<td>USA</td>
<td>DP</td>
</tr>
<tr>
<td>Electronic Business XML</td>
<td>International</td>
<td>ebXML</td>
</tr>
<tr>
<td>Electronic Data Interchange For Administration Commerce And Transport</td>
<td>International</td>
<td>EDIFACT</td>
</tr>
<tr>
<td>Electronic Data Processing</td>
<td>International</td>
<td>EDP</td>
</tr>
<tr>
<td>Electronic Funds Transfer</td>
<td>International</td>
<td>EFT</td>
</tr>
<tr>
<td>Employment Practices Liability Insurance</td>
<td>USA</td>
<td>EPLI</td>
</tr>
<tr>
<td>Equal Employment Opportunity Act</td>
<td>USA</td>
<td>EEO</td>
</tr>
<tr>
<td>Extended Personal Injury Protection</td>
<td>USA</td>
<td>EPIP</td>
</tr>
<tr>
<td>eXtensible Business Reporting Language</td>
<td>International</td>
<td>XBRL</td>
</tr>
<tr>
<td>eXtensible Markup Language</td>
<td>International</td>
<td>XML</td>
</tr>
<tr>
<td>Exterior Insulating and Finishing System</td>
<td>USA</td>
<td>EIFS</td>
</tr>
<tr>
<td>Fair Access to Insurance Requirements</td>
<td>USA</td>
<td>FAIR</td>
</tr>
<tr>
<td>Farmers Home Mortgage Administration</td>
<td>USA</td>
<td>FHMA</td>
</tr>
<tr>
<td>Expended term</td>
<td>Source</td>
<td>Acronym</td>
</tr>
<tr>
<td>------------------------------------------------------------------</td>
<td>-----------------</td>
<td>---------</td>
</tr>
<tr>
<td>Federal Emergency Management Association</td>
<td>USA</td>
<td>FEMA</td>
</tr>
<tr>
<td>Federal Employers Liability Act</td>
<td>USA</td>
<td>FELA</td>
</tr>
<tr>
<td>Financial Information eXchange Protocol</td>
<td>International</td>
<td>FIX</td>
</tr>
<tr>
<td>Financial Products Markup Language</td>
<td>International</td>
<td>FpML</td>
</tr>
<tr>
<td>Free On Board</td>
<td>USA</td>
<td>FOB</td>
</tr>
<tr>
<td>Frequency Modulation</td>
<td>International</td>
<td>FM</td>
</tr>
<tr>
<td>Health Level 7</td>
<td>International</td>
<td>HL7</td>
</tr>
<tr>
<td>Heating Ventilation And Air Conditioning</td>
<td>International</td>
<td>HVAC</td>
</tr>
<tr>
<td>Homeowners (ISOUS Forms)</td>
<td>USA</td>
<td>HO</td>
</tr>
<tr>
<td>Human Resources XML</td>
<td>International</td>
<td>HR-XML</td>
</tr>
<tr>
<td>Individual Case Report</td>
<td>USA</td>
<td>ICR</td>
</tr>
<tr>
<td>Insurance Data Management Association</td>
<td>USA</td>
<td>IDMA</td>
</tr>
<tr>
<td>Insurance Industry Internet Task Force</td>
<td>International</td>
<td>I3STF</td>
</tr>
<tr>
<td>Insurance Information Centre of Canada</td>
<td>International</td>
<td>IICC</td>
</tr>
<tr>
<td>Insurance Services Office</td>
<td>USA</td>
<td>ISOUS</td>
</tr>
<tr>
<td>Interactive Financial Exchange Forum</td>
<td>International</td>
<td>IFXForum</td>
</tr>
<tr>
<td>International Association of Industrial Accident Boards and Commissions</td>
<td>International</td>
<td>IAIABC</td>
</tr>
<tr>
<td>International Classification of Diseases</td>
<td>International</td>
<td>ICD</td>
</tr>
<tr>
<td>International Organization for Standardization</td>
<td>International</td>
<td>ISO</td>
</tr>
<tr>
<td>Interstate Commerce Commission</td>
<td>USA</td>
<td>ICC</td>
</tr>
<tr>
<td>Intravenous</td>
<td>International</td>
<td>IV</td>
</tr>
<tr>
<td>Life Office of South Africa</td>
<td>International</td>
<td>LOA</td>
</tr>
<tr>
<td>Limited Liability Company</td>
<td>International</td>
<td>LLC</td>
</tr>
<tr>
<td>Managing General Agent</td>
<td>USA</td>
<td>MGA</td>
</tr>
<tr>
<td>Medical Information Bureau</td>
<td>International</td>
<td>MIB</td>
</tr>
<tr>
<td>Minnesota Workers’ Compensation Insurers Association</td>
<td>USA</td>
<td>MWClA</td>
</tr>
<tr>
<td>Mortgage Industry Data Standards Maintenance Organization</td>
<td>International</td>
<td>MISMO</td>
</tr>
<tr>
<td>Motor Vehicle Report</td>
<td>USA</td>
<td>MVR</td>
</tr>
<tr>
<td>Multipurpose Internet Mail Extensions</td>
<td>International</td>
<td>MIME</td>
</tr>
<tr>
<td>National Association of Insurance Commissioners</td>
<td>USA</td>
<td>NAIC</td>
</tr>
<tr>
<td>North American Industry Classification System</td>
<td>North America</td>
<td>NAICS</td>
</tr>
<tr>
<td>National Association of Life Brokerage Agencies</td>
<td>USA</td>
<td>NAILBA</td>
</tr>
<tr>
<td>National Association of Securities Dealers Automated Quotation System</td>
<td>International</td>
<td>NASDAQ</td>
</tr>
<tr>
<td>National Association of Variable Annuities Society of Actuaries</td>
<td>USA</td>
<td>NAVA</td>
</tr>
<tr>
<td>National Automated Clearinghouse Association</td>
<td>International</td>
<td>NACHA</td>
</tr>
<tr>
<td>National Council on Compensation Insurance</td>
<td>USA</td>
<td>NCCI</td>
</tr>
<tr>
<td>National Crime Information Center</td>
<td>USA</td>
<td>NCIC</td>
</tr>
<tr>
<td>National Insurance Crime Bureau</td>
<td>USA</td>
<td>NICB</td>
</tr>
<tr>
<td>New Jersey Compensation Insurance Rating Board</td>
<td>USA</td>
<td>NJCIRB</td>
</tr>
<tr>
<td>New York Compensation Insurance Rating Board</td>
<td>USA</td>
<td>NYCIRB</td>
</tr>
<tr>
<td>New York Stock Exchange</td>
<td>USA</td>
<td>NYSE</td>
</tr>
<tr>
<td>North Carolina Rate Bureau</td>
<td>USA</td>
<td>NCRB</td>
</tr>
<tr>
<td>Occupational Safety and Health Administration</td>
<td>USA</td>
<td>OSHA</td>
</tr>
<tr>
<td>Expended term</td>
<td>Source</td>
<td>Acronym</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>--------------</td>
<td>---------</td>
</tr>
<tr>
<td>Open Financial Exchange</td>
<td>International</td>
<td>OFX</td>
</tr>
<tr>
<td>Owners, Landlords &amp; Tenants</td>
<td>USA</td>
<td>OLT</td>
</tr>
<tr>
<td>Pennsylvania Compensation Rating Bureau</td>
<td>USA</td>
<td>PCRB</td>
</tr>
<tr>
<td>Personal Injury Protection</td>
<td>USA</td>
<td>PIP</td>
</tr>
<tr>
<td>Polychlorinated Biphenyls</td>
<td>International</td>
<td>PCB</td>
</tr>
<tr>
<td>Polyvinyl Chloride</td>
<td>International</td>
<td>PVC</td>
</tr>
<tr>
<td>Pregnant Mare Urine</td>
<td>International</td>
<td>PMU</td>
</tr>
<tr>
<td>Public Risk DataBase</td>
<td>USA</td>
<td>PRDP</td>
</tr>
<tr>
<td>Public Safety Commission</td>
<td>USA</td>
<td>PSC</td>
</tr>
<tr>
<td>Public Utilities Commission</td>
<td>USA</td>
<td>PUC</td>
</tr>
<tr>
<td>Public SafetyCommission</td>
<td>USA</td>
<td>PSC</td>
</tr>
<tr>
<td>Radio Detection And Ranging</td>
<td>International</td>
<td>RADAR</td>
</tr>
<tr>
<td>Safe Manufacturers National Association</td>
<td>USA</td>
<td>SMNA</td>
</tr>
<tr>
<td>Securities And Exchange Commission</td>
<td>USA</td>
<td>SEC</td>
</tr>
<tr>
<td>Small Business Administration</td>
<td>USA</td>
<td>SBA</td>
</tr>
<tr>
<td>Society for Worldwide Interbank Financial Telecommunication</td>
<td>International</td>
<td>SWIFT</td>
</tr>
<tr>
<td>Standard Industry Classification</td>
<td>USA</td>
<td>SIC</td>
</tr>
<tr>
<td>Surety Association of America</td>
<td>USA</td>
<td>SAA</td>
</tr>
<tr>
<td>Television</td>
<td>International</td>
<td>TV</td>
</tr>
<tr>
<td>Terrorism Risk Insurance Act</td>
<td>USA</td>
<td>TRIA</td>
</tr>
<tr>
<td>Third Party Administrator</td>
<td>USA</td>
<td>TPA</td>
</tr>
<tr>
<td>Underwriters Laboratory</td>
<td>International</td>
<td>UL</td>
</tr>
<tr>
<td>Underwriters Laboratory Of Canada</td>
<td>International</td>
<td>ULC</td>
</tr>
<tr>
<td>United Nations</td>
<td>International</td>
<td>UN</td>
</tr>
<tr>
<td>United States Longshore and Harbor</td>
<td>USA</td>
<td>USLH</td>
</tr>
<tr>
<td>United States Of America</td>
<td>International</td>
<td>USA</td>
</tr>
<tr>
<td>Vehicle Identification Number</td>
<td>USA</td>
<td>VIN</td>
</tr>
<tr>
<td>Vehicle Information Centre Of Canada</td>
<td>International</td>
<td>VICC</td>
</tr>
<tr>
<td>Vice President</td>
<td>International</td>
<td>VP</td>
</tr>
<tr>
<td>Western European EDIFACT Group for Insurance</td>
<td>International</td>
<td>eEg7</td>
</tr>
<tr>
<td>Wisconsin Compensation Rating Bureau</td>
<td>USA</td>
<td>WCRB</td>
</tr>
<tr>
<td>Workers Compensation Insurance Rating Bureau</td>
<td>USA</td>
<td>WCIRB</td>
</tr>
<tr>
<td>Workers’ Compensation Rating and Inspection Bureau</td>
<td>USA</td>
<td>WCRIB</td>
</tr>
</tbody>
</table>
B. LIST OF CORE AND DERIVED DATATYPES

The following extract of the actual BusinessDataTypes schema module presents the detailed definitions of the ACORD Core Data Types, and of the initial set of derived Business Data Types, as presented in Chapter 4. This is followed by a sample of the remaining content of the schema module.

<?xml version="1.0" encoding="UTF-8"?>
<!-- NON NORMATIVE EXAMPLE PROVIDED FOR ACORD NDR ILLUSTRATION -->

Standard Organization: ACORD
Schema version: 1.0.0
Schema date: 12 November 2007

COPYRIGHT NOTICE:
(c) 2001-2007 ACORD. All Rights Reserved.
<!-- or directly from an ACORD Core Data Type or from one of the xsd data types authorized in the
ACORD Core Data Type list -->

<!-- ========= -->

<!-- ========= xsd:schema Element With Namespaces Declarations ========= -->

<!-- ========= -->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmns="http://www.acord.org/schema/data/standard/ReusableDataComponents/1" xmlns:acord-
doc="http://www.acord.org/schema/documentation/standard/Annotations/1"
targetNamespace="http://www.acord.org/schema/data/standard/ReusableDataComponents/1"
 elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0">
  <xsd:annotation>
    <xsd:appinfo>
      <acord-doc:GenerationDateTime>2007-11-12T18:00:00Z</acord-
doc:GenerationDateTime>
      </xsd:appinfo>
    </xsd:annotation>
    <!-- ========= -->
    <!-- ========= Imports ========= -->
    <!-- ========= -->
    <!-- ========= Unrestricted xsd types authorized for reuse as Core and Business Data Types
======-->
<!--
code, dateTime, time, gYear, gYearMonth, gMonth, gMonthDay, gDay, duration, anyURI, boolean, base64Binary-->

<!--
Unrestricted ACORD Core Data Types for reuse authorized as business data types
-->

<xsd:complexType name="Amount_Type">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="currencyCode" type="xsd:token" use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="Code_Type">
  <xsd:simpleContent>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="Measure_Type">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="unitCode" type="xsd:token"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="Numeric_Type">
  <xsd:restriction base="xsd:decimal"/>
</xsd:simpleType>

<xsd:simpleType name="Percent_Type">
  <xsd:restriction base="xsd:decimal"/>
</xsd:simpleType>

<xsd:simpleType name="UUID_Type">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="36"/>
    <xsd:minLength value="36"/>
    <xsd:pattern value="[0-9A-Fa-f]{8}(-[0-9A-Fa-f]{4}){3}-[0-9A-Fa-f]{12}"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="Identifier_Type" abstract="true">
    <xsd:simpleContent>
        <xsd:extension base="xsd:normalizedString"/>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="Quantity_Type" abstract="true">
    <xsd:simpleContent>
        <xsd:extension base="xsd:decimal">
            <xsd:attribute name="unitCode" type="xsd:token"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="Rate_Type" abstract="true">
    <xsd:simpleContent>
        <xsd:extension base="xsd:decimal"/>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="Text_Type" abstract="true">
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute ref="xml:lang"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="ACORDClosedCode_Type">
    <xsd:simpleContent>
        <xsd:extension base="xsd:QName">
            <xsd:attributeGroup ref="ClosedCodeAttribute_Sequence"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="ACORDOpenCode_Type">
    <xsd:simpleContent>
        <xsd:extension base="xsd:QName">
            <xsd:attributeGroup ref="CodeAttribute_Sequence"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="AssignedIdentifier_Type">
    <xsd:simpleContent>
        <xsd:restriction base="Identifier_Type">
            <xsd:minLength value="1"/>
            <xsd:maxLength value="255"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="SimpleIdentifier_Type"/>
<xsd:restriction base="xsd:normalizedString">
  <xsd:minLength value="1"/>
  <xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="IntegerNumeric_Type">
  <xsd:restriction base="xsd:int"/>
</xsd:simpleType>
<xsd:simpleType name="UnsignedIntegerNumeric_Type">
  <xsd:restriction base="xsd:unsignedInt"/>
</xsd:simpleType>
<xsd:simpleType name="DegreeLatitudeNumeric_Type">
  <xsd:restriction base="Numeric_Type">
    <xsd:minInclusive value="-90"/>
    <xsd:maxInclusive value="90"/>
    <xsd:totalDigits value="12"/>
    <xsd:totalDigits value="8"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="DegreeLongitudeNumeric_Type">
  <xsd:restriction base="Numeric_Type">
    <xsd:minInclusive value="-180"/>
    <xsd:maxInclusive value="180"/>
    <xsd:totalDigits value="13"/>
    <xsd:totalDigits value="8"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="SimpleRate_Type">
    <xsd:restriction base="xsd:decimal"/>
</xsd:simpleType>

<xsd:simpleType name="InfiniteText_Type">
    <xsd:restriction base="xsd:string">
        <xsd:minLength value="1"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="InfiniteLanguageText_Type">
    <xsd:simpleContent>
        <xsd:restriction base="Text_Type">
            <xsd:minLength value="1"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="LongText_Type">
    <xsd:restriction base="xsd:string">
        <xsd:minLength value="1"/>
        <xsd:maxLength value="255"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="LongLanguageText_Type">
    <xsd:simpleContent>
        <xsd:restriction base="Text_Type">
            <xsd:minLength value="1"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:maxLength value="255"/>

</xsd:restriction>
</xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="PhoneNumberText_Type">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="32"/>
    <xsd:minLength value="1"/>
    <xsd:pattern value="\+\S+\-\S+\-\S+\(\+\S+\)?"/>
  </xsd:restriction>
</xsd:simpleType>

<!-- ====== -->
<!--==== Sample Identifier scheme Types ====-->
<!-- ====== -->

<xsd:complexType name="FEINIdentifier_Type">
  <xsd:simpleContent>
    <xsd:restriction base="Identifier_Type"/>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="SocialSecurityNumberIdentifier_Type">
  <xsd:simpleContent>
    <xsd:restriction base="Identifier_Type"/>
  </xsd:simpleContent>
</xsd:complexType>

<!-- ====== -->
<!--==== Sample Code List Types ====-->
<!-- ====== -->

<xsd:complexType name="AMBestCode_Type">
  <xsd:simpleContent>
    <xsd:extension base="Code_Type"/>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="BeneficiaryDependencyCode_Type">
  <xsd:simpleContent>
    <xsd:extension base="BeneficiaryDependencyCodeContent_Type">
      <xsd:attributeGroup ref="CodeAttribute_Sequence"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="CountryCode_Type">
  <xsd:simpleContent>
    <xsd:extension base="clm53166:CountryCodeContentType">
      <xsd:attributeGroup ref="CodeAttribute_Sequence"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="CurrencyCode_Type">
  <xsd:simpleContent>
    <xsd:extension base="clm54217:CurrencyCodeContentType">
      <xsd:attributeGroup ref="CodeAttribute_Sequence"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="MaritalStatusCode_Type">
    <xsd:simpleContent>
        <xsd:extension base="MaritalStatusCodeContent_Type">
            <xsd:attributeGroup ref="ClosedCodeAttribute_Sequence"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="PostalCode_Type">
    <xsd:simpleContent>
        <xsd:extension base="Code_Type"/>
    </xsd:simpleContent>
</xsd:complexType>

<!-- ====== -->
<!--==== Abstract aggregate root type ====-->
<!-- ====== -->

<xsd:complexType name="ACORDAggregate_Type">
    <xsd:attribute name="key" type="SimpleIdentifier_Type"/>
</xsd:complexType>

<!-- ===== Attribute groups ===== -->
<!-- ====== -->

<xsd:attributeGroup name="CodeAttribute_Sequence">
    <xsd:attribute name="name" type="xsd:normalizedString" use="optional"/>
    <xsd:attribute ref="xml:lang" use="optional"/>
    <xsd:attribute name="codeListReference" type="SimpleIdentifier_Type" use="optional"/>
</xsd:attributeGroup>
</xsd:attributeGroup>

<xsd:attributeGroup name="ClosedCodeAttribute_Sequence">
  
  <xsd:attribute name="name" type="xsd:normalizedString" use="optional"/>

  <xsd:attribute ref="xml:lang" use="optional"/>

</xsd:attributeGroup>

</xsd:schema>
C. SCHEMA MODULE TEMPLATES

FUTURE ADDITION
D. ANNOTATION TEMPLATES

Schema Element

```xml
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.acord.org/schema/data/draft/ReusableDataComponents/1"
xmlns:acord-doc="http://www.acord.org/schema/documentation/draft/Annotations/1"
targetNamespace="http://www.acord.org/schema/data/draft/ReusableDataComponents/1"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0">
  <xsd:annotation>
    <xsd:appinfo>
      <acord-doc:GenerationDateTime>2007-08-17T18:06:25Z</acord-doc:GenerationDateTime>
    </xsd:appinfo>
  </xsd:annotation>

Simple Type

```xml
<xsd:simpleType name="AboveGroundIndicator_Type">
  <xsd:annotation>
    <xsd:appinfo>
      <acord-doc:ComponentVersion>1.0.0</acord-doc:ComponentVersion>
    </xsd:appinfo>
    <xsd:documentation>
      <acord-doc:ExpandedName>Above Ground Indicator Type</acord-doc:ExpandedName>
      <acord-doc:Description>
        <acord-doc:p>An indication of whether the item is above ground.</acord-doc:p>
      </acord-doc:Description>
    </xsd:documentation>
  </xsd:annotation>
</xsd:simpleType>
```
Global Element

<xsd:element name="Organization" type="Organization_Type">
   <xsd:annotation>
      <xsd:documentation>
         <acord-doc:BusinessTerm>Organization</acord-doc:BusinessTerm>
         <acord-doc:Description>
            <acord-doc:p>This contains data about a company.</acord-doc:p>
            </acord-doc:Description>
            </xsd:documentation>
            <xsd:appinfo>
               <acord-doc:ComponentVersion>1.0.0</acord-doc:ComponentVersion>
               </xsd:appinfo>
   </xsd:annotation>
</xsd:element>

Complex Type and Contained Local Elements

<xsd:complexType name="Organization_Type">
   <xsd:annotation>
      <xsd:documentation>
         <acord-doc:ExpandedName>Organization Type</acord-doc:ExpandedName>
         <acord-doc:Description>
            <acord-doc:p>This contains data about a company.</acord-doc:p>
            </acord-doc:Description>
         </xsd:documentation>
      </xsd:annotation>
</xsd:complexType>
Complex Type and Contained Global Elements

<xsd:complexType name="Claim_Type">

</xsd:complexType>
<xsd:complexType name="Claim_Type">
  <xsd:annotation>
    <xsd:documentation>
      <acord-doc:ExpandedName>Claim Type</acord-doc:ExpandedName>
      <acord-doc:Description>
        This contains data about a claim.
      </acord-doc:Description>
    </xsd:documentation>
    <xsd:appinfo>
      <acord-doc:ComponentVersion>1.0.0</acord-doc:ComponentVersion>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element ref="AdditionalInterest" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Beneficiary" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

Complex Type and Referenced Global Elements
<acord-doc:KeySelector>ac:AdditionalInterest</acord-doc:KeySelector>
<acord-doc:KeySelector>ac:Beneficiary</acord-doc:KeySelector>

</xsd:annotation>
<xsd:sequence>
  <xsd:element ref="AdditionalInterest" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="Beneficiary" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>

**Code Lists and Declared Enumerations**

<xsd:simpleType name="MaritalStatusCode_List" >
  <xsd:annotation>
    <xsd:documentation>
      <acord-doc:ExpandedName>Marital Status Code List</acord-doc:ExpandedName>
      <acord-doc:Description>
        <acord-doc:p>A code indicating the present marital status of the individual.</acord-doc:p>
      </acord-doc:Description>
    </xsd:documentation>
  </xsd:annotation>
  <xsd:appinfo>
    <acord-doc:ComponentVersion>1.0.0</acord-doc:ComponentVersion>
  </xsd:appinfo>
</xsd:simpleType>
<acord-doc:ExpandedName>Divorced</acord-doc:ExpandedName>

<acord-doc:Description>

<acord-doc:p>Divorced</acord-doc:p>

</acord-doc:Description>

</xsd:documentation>

</xsd:annotation>

</xsd:enumeration>

.....

</xsd:restriction>

</xsd:simpleType>
E. VERSIONING POLICY

Background

What is Compatibility

Compatibility is defined in a manner that centers upon the XML instance documents that result from or are compliant with the use of ACORD Standards. Schema changes or constructs that do not require changes in the instance documents are of no concern with regard to compatibility.

Defining Compatibility requires that one examine the standard from the perspective of a single version structurally, and then describe both backward compatibility and forward compatibility as positions relative to the specific structural version. The key is the structure of the message.

From the point of view of a standard’s evolution, a big disadvantage to maintaining Compatibility is that, often times, in order to add new functionality while protecting what exists, the standard becomes highly redundant and the underlying model can easily become unmanageable. As a result, the added value of Compatibility is a key feature in deciding what the policy will be within a standard.

Backward Compatibility

Backward compatibility is relevant when the receiver or consumer of a message is expecting a later version of the ACORD Standard than the requestor is creating or transmitting. The underlying assumption of being backward compatible is that the service being invoked should be in sync with or ahead of, in terms of version, the systems being integrated with it.

Backward compatibility demands that the semantics of an instance document remain consistent from the version in which it is first implemented through all future versions within the same major release series (1.x, 2.x, etc.) without need of trading partner negotiations or code changes.

Backward compatibility demands that the structure of the standard, including data types, element order, and enumerations must remain consistent from one version to a subsequent version within the same major release series (1.x, 2.x, etc.) for all elements of the standard present in the earlier version. Changes would only be permitted if they do not conflict with prior structural definitions.

Backward compatibility does NOT prohibit the addition of new elements or functionality to the standard in subsequent versions within the same major release series (1.x, 2.x, etc.), so long as such additions do not change the interpretation of elements that exist in an earlier version of the standard.

Backward compatibility does NOT mandate that the receiver must remain current with the latest published release of the standards. The expectation is that the receiver
would be updated when it or one of its clients had need of additional functionality in the later version.

**Forward Compatibility**

Forward compatibility is relevant when the requestor of a message is using a later version of the ACORD standard than the receiver or consumer is expecting or is capable of processing.

Forward compatibility allows the requestors to upgrade their system without regard to what version of the standard the receivers are compliant with. Forward Compatibility is rare in the IT industry, and in fact, prior to the emergence of XML, it was rarely - if ever - discussed.

Forward compatibility can be accomplished because new tags in XML can be invisible to systems that don’t specifically look for them. One approach to enable this type of compatibility is to allow the receiver of a message to ignore all new tags that it does not recognize. In theory then, so long as new additions are made only through the addition of new tags and do not effect the interpretation of the current tags, once a service is written to support a specific version of the standard, it will never need to be changed except to make use of additional functionality in a later version. In terms of allowing systems to continue to function without being upgraded, this approach would work. However, this approach is not desirable because if a receiver ignores structural elements of a message, there is no way that the receiver or the requestor can know that the entire intent of the requestor was accomplished.

Forward compatibility then must take the approach that, regardless of the version number of the message being sent, it is the responsibility of the requestor to not include elements in a message that are newer than the version of the standard the receiver is currently supporting.

Additionally, it is the responsibility of the receiver to ensure that it “understands” all structural elements that are being sent.

**Compatibility Policy**

- **Backward Compatibility**, as it is defined in this document, SHALL BE maintained between versions of the ACORD Standards within the same major release series (1.x, 2.x, etc.).

- **Forward Compatibility** SHALL NOT BE guaranteed or intentionally maintained in the ACORD Standards. Members are not specifically encouraged to seek Forward Compatibility due to the risk of a receiver processing a message without considering its full business intent. Typically, a message would only be found to be Forward Compatible when there have been no changes to it from one version to a subsequent version.
Policy Implications

The sections that follow identify the requirements or responsibilities that result from this policy for the ACORD Standards constituents.

Responsibilities of Implementers

While Compatibility has traditionally defined how ACORD managed the evolution of the standard, as discussed previously, Compatibility can only be insured if the implementers (requestors and receivers) follow specific rules in regard to their behavior.

- An ACORD message should always contain the version number as an attribute on the root level element of the instance document as expressed in the standard documentation. A current underlying assumption is that trading partners will determine what to do based upon the relationship between the received version number and the version(s) they support.

- It is the responsibility of the message creator to not include elements in a message that are newer than the version of the standard indicated in the version attribute on the message root element.

- It is NOT mandated that a receiver must remain current with the latest published version of the standards. Senders MAY expect that a receiver would be updated when it or one of its clients had need of additional functionality in the later version, and best practice dictates that anyone integrated with the service would be properly notified of any impact.

Membership Responsibilities for Evolution

This section defines the rules that guide the evolution of the ACORD Standard from one version to the next. Careful evolution of the standard will enable Compatibility. It is expected that these rules will be used by the ACORD membership in evaluating and voting on Maintenance Requests. These policies are not intended to block the submission of Maintenance Requests to the standard.

- The semantics of structural elements must remain consistent from the version in which they are first implemented through all future versions. No Maintenance Request will be permitted to change the semantics of an existing object, property, or value in the ACORD Standards in a way that will create ambiguity or change the interpretation of the item in existing implementations. This includes updates to the item itself and the addition of any other items that may impact its meaning.

- The structure of the existing elements in the standard, including element names, data types, element order, and enumerations must remain consistent from one version through all subsequent versions. Changes would only be permitted if they do not conflict with prior definitions, and have no impact on the validation of any existing instance documents or implementations.

  - The only acceptable exception is in the case where the implementation of the element was implemented in the standard was flawed, and an error correction was required.
• This requirement does not restrict the placement of new elements in the sequence in the schema.

☐ New elements must not be added, if their addition changes the semantic meaning of existing elements.

☐ While the creation of redundant constructs in the standard should be avoided as much as possible, when new elements or functionality are added to the standard in subsequent versions (within the same major release series) that result in a redundancy with an existing mechanism for capturing equivalent information, the redundancy will be documented on both the new and existing item(s) while a solution\(^1\) to the redundancy will be explicitly listed in the documentation associated with the change.

☐ Deprecation of an element, structure, or code, will be permitted under certain circumstances, such as following the introduction of redundant elements as explained above. Deprecations will be treated as a two-step process. The item is first deprecated by making a change to the specification noting that the item should no longer be used and will be deleted. With a second vote to approve the deletion, the deprecated item will be removed from the specification.

### Categorization of Maintenance Requests

**Definitions of the change severity codes:**

‘Major’: indicates that the specification is modified to a degree that new compliant XML instance documents may not be backward compatible with instance documents based on older versions.

‘Minor’: indicates that the specification is modified in a way that new compliant XML instance documents will still be backward compatible with instance documents based on older versions.

‘Revision’: indicates that the specification is not modified other than correcting errors or enhancing clarity.

<table>
<thead>
<tr>
<th>Category</th>
<th>Change Type</th>
<th>Change severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element or attribute</td>
<td>Change tag description. This involves a change such as correcting a typographical error or revising wording for clarity.</td>
<td>Revision</td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Change tag description/meaning. This involves a change such as changing the</td>
<td>Major</td>
</tr>
</tbody>
</table>

\(^1\) The solution to redundancy can include a requirement that the newer mechanism must be supported beyond a specified version, duplicity/controlled redundancy, deprecation, context-specific processing rules, or that both mechanisms must be supported beyond a certain version. The specifics of the solution are expected to vary depending on the redundancy being created.
<table>
<thead>
<tr>
<th>Category</th>
<th>Change Type</th>
<th>Change severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>semantic meaning of the tag.</td>
<td></td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Add when usage is required</td>
<td>Major</td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Add when usage is optional</td>
<td>Minor</td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Add redundant tag when usage is optional</td>
<td>Minor^1</td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Delete</td>
<td>Major</td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Change Name</td>
<td>Major</td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Moving tag to another namespace.</td>
<td>Major</td>
</tr>
<tr>
<td>Element</td>
<td>Move within parent tag (re-sort tag order).</td>
<td>Major</td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Change usage from Required to Optional</td>
<td>Minor</td>
</tr>
<tr>
<td>Element</td>
<td>Change usage from Optional to Optional Repeating</td>
<td>Minor</td>
</tr>
<tr>
<td>Element</td>
<td>Change Required to Required Repeating</td>
<td>Minor</td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Change usage from Optional to Required</td>
<td>Major</td>
</tr>
<tr>
<td>Element</td>
<td>Change cardinality to a higher minimum value</td>
<td>Major</td>
</tr>
<tr>
<td>Element</td>
<td>Change cardinality to a lower maximum</td>
<td>Major</td>
</tr>
</tbody>
</table>

^1 Note: this affects forward compatibility and should be avoided.
<table>
<thead>
<tr>
<th>Category</th>
<th>Change Type</th>
<th>Change severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Change description of tag to add a redundant code list that is mutually exclusive to the specified code list.</td>
<td>Minor³</td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Change description of tag to point to different code list.</td>
<td>Major</td>
</tr>
<tr>
<td>Element or attribute</td>
<td>Change Data Type of element’s or attribute’s simple content</td>
<td>Major or Minor  (see Change Type notes)</td>
</tr>
<tr>
<td></td>
<td>Any change of a data type that allows the tag to store a superset of its previously legal values is considered a Minor change, examples are:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Extending string length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Closed enum to an open enum</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Adding values to a closed or open enum</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Long to Decimal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Changing a Numeric or specific data type such as PhoneNumber or DateTime to a String type is not allowed as a Minor change because this may change the interpretation of an already existing value thereby violating the Superset rule.</td>
<td></td>
</tr>
<tr>
<td>Data Type</td>
<td>Adding a new data type definition.</td>
<td>Minor</td>
</tr>
<tr>
<td>Data Type</td>
<td>Changing the definition of a simple content data type.</td>
<td>Major</td>
</tr>
<tr>
<td></td>
<td>Note: To change the Data Type of an element’s or attribute’s simple content, a new data type should be defined</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Change Type</td>
<td>Change severity</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Data Type</td>
<td>Deleting a data type definition.</td>
<td>Major</td>
</tr>
<tr>
<td>Code</td>
<td>Add a Code to an Enumeration</td>
<td>Minor</td>
</tr>
<tr>
<td>Code</td>
<td>Add a redundant Code to an Enumeration</td>
<td>Minor¹</td>
</tr>
<tr>
<td>Code</td>
<td>Change Code Description</td>
<td>Revision</td>
</tr>
<tr>
<td></td>
<td>- when the change does not impact the semantic meaning of the code.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ex. A = Apples change to A = Apples</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Change Code Description/ Meaning</td>
<td>Major</td>
</tr>
<tr>
<td></td>
<td>- when the change does impact the semantic meaning of the code.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ex. A = Apples is changed to A = Apricots.</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Delete a Code from an Enumeration</td>
<td>Major</td>
</tr>
<tr>
<td>Code</td>
<td>Change a Code value</td>
<td>Major</td>
</tr>
<tr>
<td></td>
<td>Ex. A = Apples is changed to 1 = Apples. This is equivalent to a delete and add.</td>
<td></td>
</tr>
<tr>
<td>Code List</td>
<td>Add a Code List (is only permitted as part of element or attribute change or addition)</td>
<td>See relevant change</td>
</tr>
<tr>
<td>Code List</td>
<td>Delete a Code List</td>
<td>Major</td>
</tr>
<tr>
<td>Code List</td>
<td>Change Code List Name</td>
<td>Major</td>
</tr>
<tr>
<td></td>
<td>Changing a Code List Name implies deleting the old list in favor of a new list name or adding the codes to another existing list. And, this would also involve deleting a code from the CodeLists list which is a Major change on its own (delete a code).</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Add a message</td>
<td>Minor</td>
</tr>
<tr>
<td>Category</td>
<td>Change Type</td>
<td>Change severity</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Function</td>
<td>Delete a message</td>
<td>Major</td>
</tr>
</tbody>
</table>
F. REQUIREMENTS FOR STANDARD METHOD FOR DESIGNING,
PUBLISHING AND MANAGING CUSTOMIZED SPECIFICATIONS

The following is an initial set of requirements to lead to the specification of a standard customization method. This determination is left for future work.

Requirements

General requirements for designing, publishing and managing customized specifications, applicable to schemas or document instances are as follows:

1. We need a controlled way of adding, removing or restricting specific elements, attributes, and codes from the original standard. ‘Controlled way’ means:
   a. Modified schema content models is documented to trading partners
   b. Customized documents can be validated by schema
   c. For example:
      When company “A” publishes a secondary schema, it should specifically define its additional content and restriction information and this information should be usable for validation of the stream when transmitted in its entirety by the trading partners.
2. We do not need to address business rules extensions comprehensively; we are only adding additional or restricting the use of certain attributes, elements, or code values.
3. We need to be able to identify the source / ownership / definition of the extensions in the schema and in the instance document.
4. At least one methodology for tight validation using schemas should exist.
5. A receiver of an instance document which contains extensions should be able to validate only the ACORD portion of the instance document with the ACORD schema. The unmodified instance should validate against the unmodified ACORD schema.
   a. A particular case is that of hubs acting as aggregators of information of various origins, containing extensions. The above rule can be met by removing the extensions but tight validation of extensions might be impractical.
6. An ACORD schema will be built in such a way that users will be able to extend the schema without having to modify the normative ACORD schema document.
7. We require that any structural modifications by an organization, once made, is easy to migrate to the next Minor version of the ACORD Standard.
8. We need a method that doesn’t require duplication of the normative schema (or element / type declarations) to define the extensions and restrictions.
   a. Users should be able to specify their modifications without having to be aware of additions by ACORD to the main schema.
b. However, in general, specifying restricted content is not possible without duplicating part of the original content; so this goal may not be fully achievable for restrictions.

9. We want a method that members and outside organizations can easily implement and is properly supported by the tools available.

10. It is desirable that extensions re-use ACORD defined aggregates as much as possible.

**Business Issues**

1. Handle company-unique content (tags, code lists, code list values) in a standard way that XML parsers will recognize.

2. Leverage the power of schema validators to define and check data in business documents, i.e. data types, code values, and structure.

**Technical Issues**

1. Works with multiple parsers & other tools (e.g. code generators) on multiple platforms; the two major platforms are specific to Microsoft and Java (MSFT has .NET and MSXML)

2. Name collision protection for code values, attributes, and elements within the defined ACORD Standard (i.e. Life / Annuity, P&C, RLC) as well as company extensions / combination of each of these (This is linked to identification of extension ownership)

3. Allow users to re-use ACORD model items (e.g. code list, aggregates) as much as possible

4. Minimize schema code needed to create the modifications