Oracle® Application Integration Architecture - Enterprise Object Library 2.5: Enterprise Business Objects and Messages XML Naming and Design Rules Guide

Release 2.5

Part No. E15765-01

October 2009

Oracle Application Integration Architecture - Enterprise Object Library 2.5: Enterprise Business Objects and Messages XML Naming and Design Rules Guide

Part No. E15765-01

Contents

Chapter 1: Introduction

The Enterprise Business Objects/Messages XML Naming and Design Rules specification (this document) defines Oracle's rules for defining business Enterprise Business Objects (EBOs) and Enterprise Business Messages (EBMs) expressed in XML [XML] and XML Schema [XMLSchema1][XMLSchema2].

This introduction discusses:

- Scope and focus
- Support for standards
- Terminology and notation
- Related documents
- Guiding Principles
- Conformance

Scope and Focus

This specification can be used wherever extensions to the Oracle schemas are to be made, in the understanding of the messages to Oracle applications, and in the design of other XML schemas for defining the EBO content model for information exchange. The scope of this document is limited to specifying rules for the library, EBOs, and EBMs.

Support for Standards

This specification is based primarily on the UN/CEFACT XML Naming and Design Rules [UNCEFACTND]. It is also based on the XML Naming and Design Rules provided by the Open Application Group [OAGINDR]. While these specifications provide a base for this specification concerning design and structure, they are of larger scope for this specification, including mapping rules, rule-by-rule profiling, and more.

Terminology and Notation

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 the Internet Engineering Task Force (IETF) Request for Comments (RFC) 2119 [Keywords].

Wherever "xsd" or "XSD" appears, it is in reference to the constructs from W3C XML schema specification [XMLSchema1]. Wherever Core Components Technical Specification (CCTS) appears, it is in reference to constructs from CCTS [CCTS].

The following notations are used throughout this document:

[Example] – A representation of a definition or rule that is intended to be informative.

[Note] – Explanatory information that is intended to be informative.

[R n] – Denotes the identification of a rule that requires conformance.

Rules are normative. To ensure continuity across versions of the specification, rule numbers that are deleted will not be reissued and any new rules will be assigned the next higher number regardless of the location.

When defining rules, this document uses the following annotations are used:

[ ] – Optional

< > – Variable

| – Choice

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

Related Documents

Related documents and the versions to this specification are defined in the reference section.

Guiding Principles

The guiding principles for this specification are to:

- Align with the UNCEFACT XML NDR [CEFACTNDR] documents and the OAGi XML NDR [OAGISNDR] documents where practically possible and in consideration of Oracle's business needs.
- Ensure the practical application of XML Schema in Oracle applications and customer installation of Oracle applications to enable business integrations today.
- Simplify the use of XML and XML Schema by providing consistency.

Conformance

Applications will be considered to be in full conformance with this specification if they comply with the rules and definitions. These rules and definitions are provided with the structure of this document.

Conformance SHALL be determined through the adherence of the rules and definitions defined within this document.

Chapter 2: AIA Terminology Usage

The following table provided a glossary of AIA-specific terminology:

Term	Definition
Components	The logical grouping of parts of an object. For example, an address is a component of a party.
Enterprise Business Object (EBO)	The large-grained business objects that are used to do business by sharing data across heterogeneous business applications both internal and external. The EBO defines all information known for the given object. An EBO instance is part of every EBM. Content of EBO instances content are affected by verbs.
Enterprise Business Message (EBM)	Messages that are passed between applications. EBMs communicate the intent of the message using the verb. The verb affects the content of the instance of the EBO.
Verb	Used to identify the intent of the EBM. The verb also affects the content of the instance of the EBO portion of the EBM.
Schema Module	Means the same as a schema file.

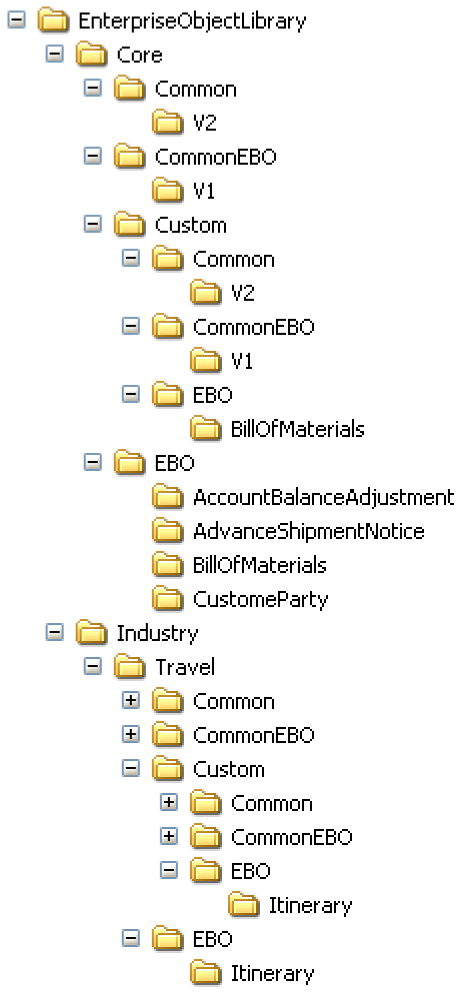
Chapter 3: Physical Packaging Structure

This section describes how the Enterprise Object Library is physically packaged and discusses:

- Industry-neutral (core) information
- Industry-specific information

Understanding the Physical Packaging Structure

The Enterprise Object Library is physically packaged as a directory structure. The following example illustrates the library. Standard UNIX directory formatting is used within the rules associated with the physical packaging of the directory.



```
EnterpriseObjectLibrary
    Core
        Common
            V2
        CommonEBO
            V1
        Custom
            Common
                V2
            CommonEBO
                V1
            EBO
                BillOfMaterials
        EBO
            AccountBalanceAdjustment
            AdvanceShipmentNotice
            BillOfMaterials
            CustomeParty
    Industry
        Travel
            Common
            CommonEBO
            Custom
                Common
                CommonEBO
                EBO
                    Itinerary
            EBO
                Itinerary
```

Figure 1: An Enterprise Object Library directory structure.

The names of all directories MUST use upper camel case.

The root directory of the Enterprise Object Library MUST be named "/EnterpriseObjectLibrary".

A library contains a Core directory, which contains all industry-neutral information, and may contain an Industry directory, which contains industry-specific information.

For all directory paths of "/EnterpriseObjectLibrary/Release<X>", one directory path of "/EnterpriseObjectLibrary/Core" MUST exist.

Industry-Neutral (Core) Information

All industry-neutral information within a library exists under the Core directory. It contains common information, common EBOs, EBOs, and a custom directory.

For all directory paths of "/EnterpriseObjectLibrary/Core", one directory path of "/EnterpriseObjectLibrary/Core/Common" MUST exist.

For all directory paths of "/EnterpriseObjectLibrary/Core", one directory path of "/EnterpriseObjectLibrary/Core/CommonEBO" MUST exist.

For all directory paths of "/EnterpriseObjectLibrary/Core", one directory path of "/EnterpriseObjectLibrary/Core/EBO" MUST exist.

For all directory paths of "/EnterpriseObjectLibrary/Core", one directory path of "/EnterpriseObjectLibrary/Core/Custom" MUST exist.

Common information is provided inside directories representing their version.

For all directory paths of "/EnterpriseObjectLibrary/Core/Common", one directory path of "/EnterpriseObjectLibrary/Core/Common/V<Y>" MUST exist, where <Y> is the major version of the Core Common information.

The Common EBOs are provided inside directories representing their version.

For all directory paths of "/EnterpriseObjectLibrary/Core/CommonEBO", one directory path of "/EnterpriseObjectLibrary/Core/CommonEBO/V<Y>" MUST exist, where <Y> is the major version of the Core Common EBOs.

Core EBOs are provided inside directories reflecting their name.

For all directory paths of "/EnterpriseObjectLibrary/Core/EBO", EBOs MUST always exist within the library using a directory path of "/EnterpriseObjectLibrary/Core/EBO/<Y>" where <Y> is the name of the EBO.

About Custom Directories

Throughout the Enterprise Object Library, special customization directories are provided for users to make extensions within the library. These directories are named "Custom," and appear in appropriate locations for extensions. They contain a set of directories mimicking the parent directory (except for itself). The Custom directories and the contained schema modules are intended to be used by customers, implementers, or both as needed.

For all directory paths of "/EnterpriseObjectLibrary/Core", one directory path of "/EnterpriseObjectLibrary/Core/Custom" MUST exist.

For all directory paths of "/EnterpriseObjectLibrary/Core/Custom", one directory path of "/EnterpriseObjectLibrary/Core/Custom/Common" MUST exist.

For all directory paths of “/EnterpriseObjectLibrary/Core/Custom/Common”, one directory path of “/EnterpriseObjectLibrary/Core/Custom/Common/V<Y>” MUST exist where <Y> is the major version of the Core Custom Common information.

For all directory paths of “/EnterpriseObjectLibrary/Core/Custom”, one directory path of “/EnterpriseObjectLibrary/Core/Custom/CommonEBO” MUST exist.

For all directory paths of “/EnterpriseObjectLibrary/Core/Custom/CommonEBO”, one directory path of “/EnterpriseObjectLibrary/Core/Custom/CommonEBO/V<Y>” MUST exist where <Y> is the major version of the Core Custom Common EBOs.

For all directory paths of “/EnterpriseObjectLibrary/Core/Custom”, one directory path of “/EnterpriseObjectLibrary/Core/Custom/EBO” MUST exist.

For all directory paths of “/EnterpriseObjectLibrary/Core/Custom/EBO”, customized EBOs MUST always exist within the library using a directory path of “/EnterpriseObjectLibrary/Core/Custom/EBO/<Y>” where <Y> is the name of the customized EBO.

Industry-Specific Information

All industry-specific information within a library exists under the Industry directory. It contains a subdirectory named according to a specific industry. Industry information is based on the core information within the library structure.

For all directory paths of “/EnterpriseObjectLibrary/Industry”, at least one directory path associated with a specific industry MUST exist of path name “/EnterpriseObjectLibrary/Industry/<Y>” where <Y> is the name of the industry.

The remaining directory structure under a release of a specific industry follows the same structure as within the core directory as specified in the pervious section.

Chapter 4: Modularity Model

This section provides a description of the Modularity Model and discusses:

- Core content modularity
- Industry content modularity
- Management modularity

Understanding the Modularity Model

Modularity of the library contents promotes reuse and enables the management of different objects, components, and their associations from an XML schema and messaging standpoint. The Modularity Model of the library defines a flexible approach and avoids monolithic brittle designs. It is enabled by both the physical directory structure of the library and the location of contents within a set of architected schema modules.

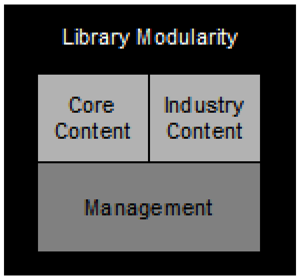


Figure 2: Modularity model dimensions

The modularity model can be thought of as existing in three dimensions: Core Content Modularity, Industry Content Modularity, and Management Modularity.

Core Content Modularity

Core Content modularity addresses the industry-neutral business information and supportive structures within the library. This section discusses:

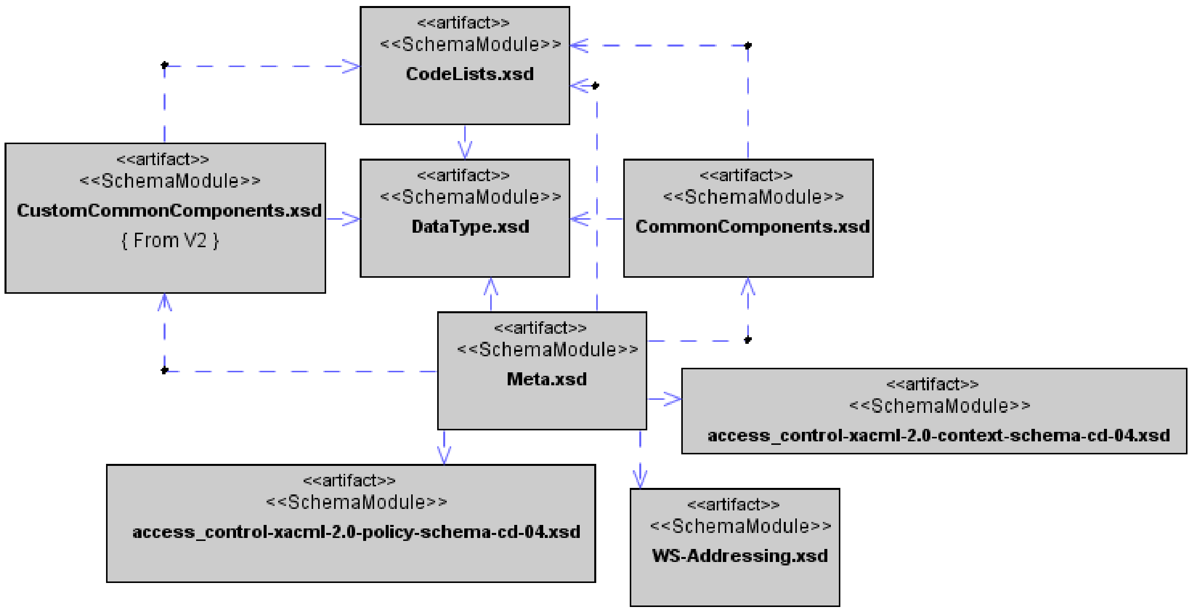
- Schema modules dependencies (core)
- Common information (core)
- Common enterprise business objects (core)
- Enterprise business objects (core)

Schema Modules Dependencies (core)

The following sections define the dependencies of the Core schema modules. A “UML as sketch” approach, which is not to define a formal UML profile, is used to depict the dependencies. A Schema Module stereotype is added to a UML artifact. The diagrams use UML dependency relationships to define the dependencies. A dependency is a relationship in which one element (the client) requires the presence of another element (the supplier) for its correct functioning or implementation. In the diagrams, the arrowhead represents the supplier.

Common Schema Module Dependencies

The following diagram defines the Common Information schema module dependencies. A set of Common schema modules exists within a specific versioned directory.



The CodeLists schema module MUST include the DataTypes schema module

The CommonComponents schema module MUST include the DataTypes schema module.

The CommonComponents schema module MUST include the CodeLists schema module.

The CommonComponents schema module MUST import the CustomCommonComponents schema module

The CustomCommonComponents schema module MUST include the DataTypes schema module

The CustomCommonComponents schema module MUST include the CodeLists schema module

The Meta schema module MUST include the DataTypes schema module.

The Meta schema module MUST include the CodeLists schema module.

The Meta schema module MUST include the CommonComponents schema module.

The Meta schema module MUST import the CustomCommonComponents schema module.

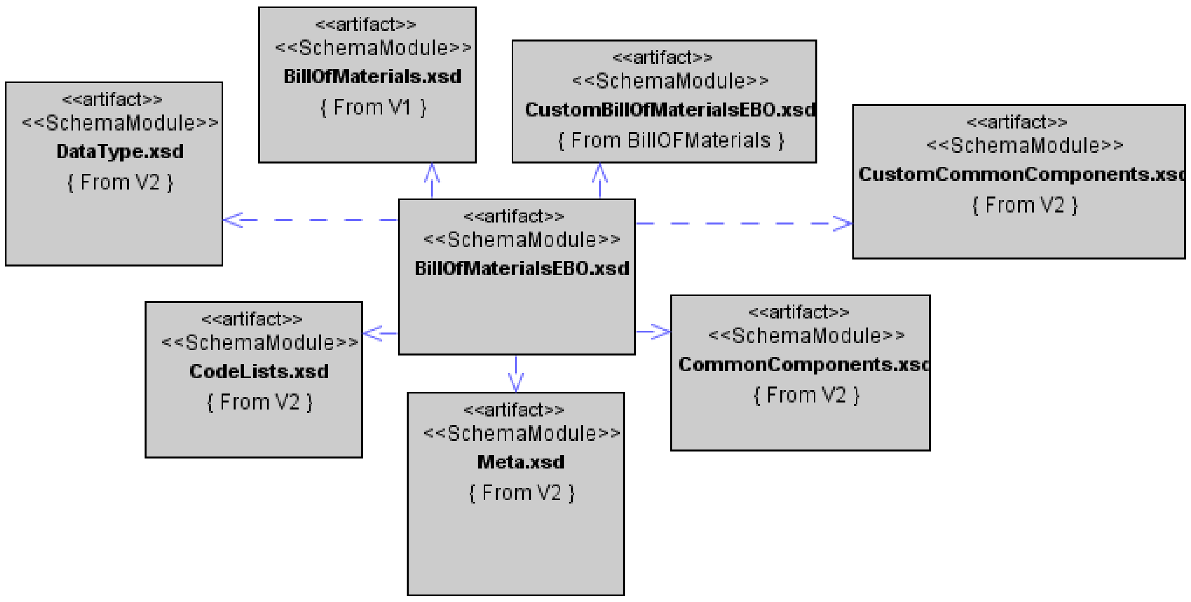
The Meta schema module MUST import the WS-Addressing schema module.

The Meta schema module MUST import the access_control-xacml-2.0-context-schema-cd-04 schema module

The Meta schema module MUST import the access_control-xacml-2.0-policy-schema-cd-04 schema module

CommonEBO Schema Module Dependencies

The following diagram exemplifies Common EBO dependencies using the BillOfMaterials.xsd CommonEBO schema module as an example.



A CommonEBO schema module MUST import the DataTypes schema module.

A CommonEBO schema module MUST import the CodeLists schema module.

A CommonEBO schema module MUST import the Meta schema module.

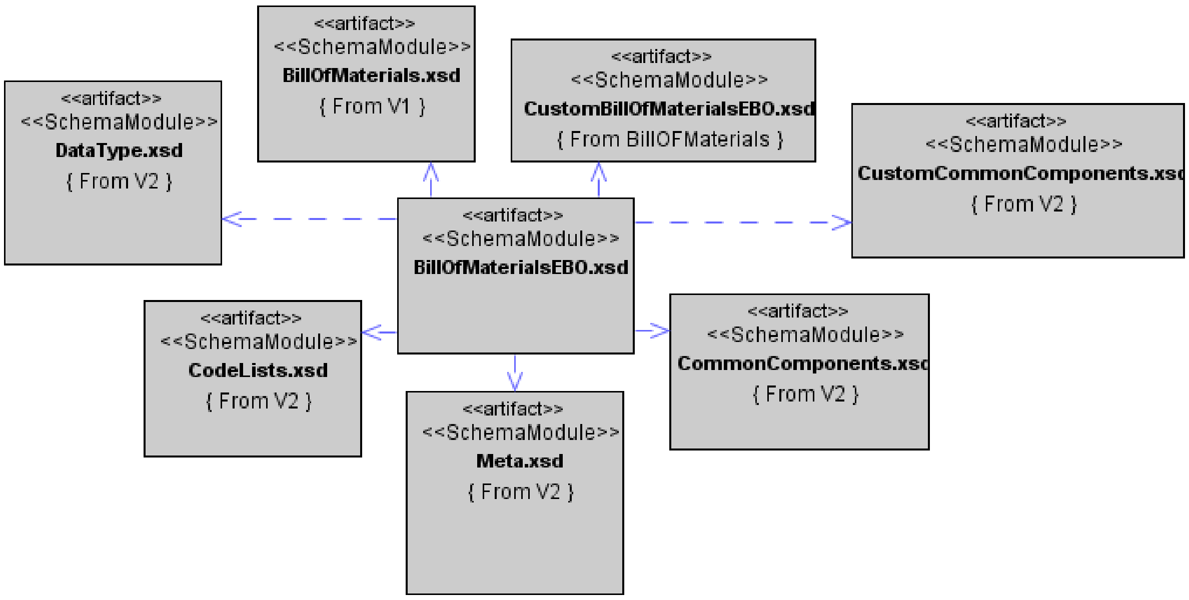
A CommonEBO schema module MUST import the CommonComponents schema module.

A CommonEBO schema module MUST import the CustomCommonComponents schema module.

A CommonEBO schema module MUST import its own custom Common EBO schema module.

EBO/EBM Schema Module Dependencies

The following diagram exemplifies EBO dependencies using the BillOfMaterialsEBO.xsd EBO schema module. EBOs are version-specific concerning importing of the Common Components.



An EBO schema module MUST import the DataTypes schema module.

An EBO schema module MUST import the CodeLists schema module.

An EBO schema module MUST import the Meta schema module.

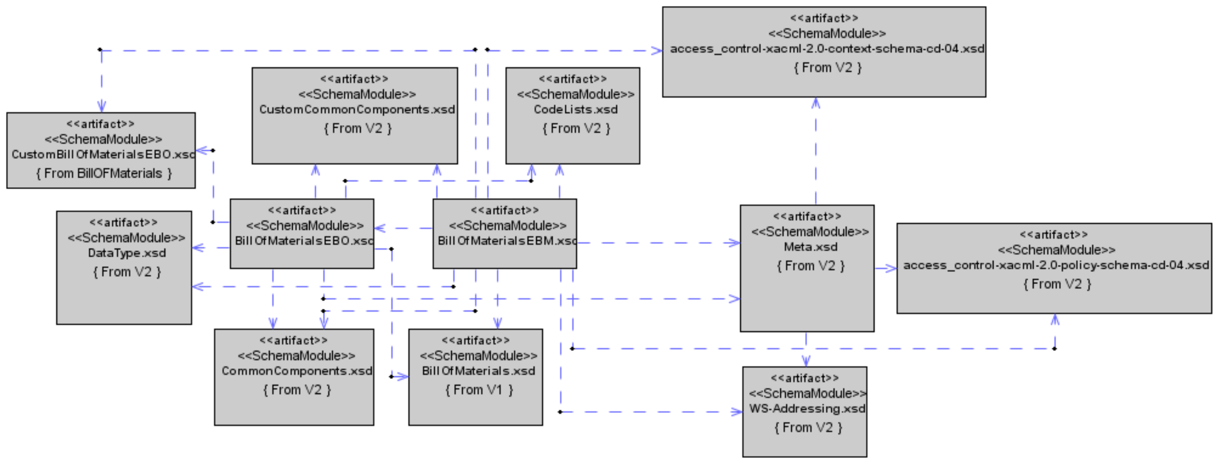
An EBO schema module MUST import the CommonComponents schema module.

An EBO schema module MUST import the CustomCommonComponents schema module.

An EBO schema module MUST import common EBO schema modules if any are required for the definition of that EBO.

An EBO schema module MUST import its Custom EBO schema module

The following diagram exemplifies EBM dependencies using the BillOfMaterialEBM.xsd schema module. EBMs are version-specific concerning importing of the Common Components.


<<artifact>> <<SchemaModule>> access_control-xacml-2.0-context-schema-cd-04.xsd { From V2 }
<<artifact>> <<SchemaModule>> CustomCommonComponents.xsd { From V2 }
<<artifact>> <<SchemaModule>> CodeLists.xsd { From V2 }
<<artifact>> <<SchemaModule>> CustomBillOfMaterialsEBO.xsd { From BillOFMaterials }
<<artifact>> <<SchemaModule>> DataType.xsd { From V2 }
<<artifact>> <<SchemaModule>> BillOfMaterialsEBO.xsd
<<artifact>> <<SchemaModule>> BillOfMaterialsEBM.xsd
<<artifact>> <<SchemaModule>> Meta.xsd { From V2 }
<<artifact>> <<SchemaModule>> access_control-xacml-2.0-policy-schema-cd-04.xsd { From V2 }
<<artifact>> <<SchemaModule>> CommonComponents.xsd { From V2 }
<<artifact>> <<SchemaModule>> BillOfMaterials.xsd { From V1 }
<<artifact>> <<SchemaModule>> WS-Addressing.xsd { From V2 }

An EBM schema module MUST import the DataTypes schema module.

An EBM schema module MUST import the CodeLists schema module.

An EBM schema module MUST import the Meta schema module.

An EBM schema module MUST import the CommonComponents schema module.

An EBM schema module MUST import the CustomCommonComponents schema module.

An EBM schema module MUST import its Common EBO schema module

An EBM schema module MUST import its Custom EBO schema module

An EBM schema module MUST import the WS-Addressing schema module

An EBM schema module MUST import the access_control-xacml-2.0-context-schema-cd-04 schema module

An EBM schema module MUST import the access_control-xacml-2.0-policy-schema-cd-04 schema module

Common Information (core)

The Schema modules for industry-neutral Common Information are provided inside versioned directories within the library. All Common Information within the library is defined within the same namespace name using the same Namespace name prefix. This allows a version to be published, which is a snapshot of the EBOs. It also allows minor version changes to be published as patches to the Common schema modules.

> The Core Common Information namespace name MUST be in the following form: http://xmlns/oracle/com/EnterpriseObjects/Core/Common/V<X> where <X> is the identification of the sequential major version number of the Core Common schema modules.
>
> The Namespace name prefix for the Core Common Information namespace MUST be “`corecom`”.

Common Information exists in the library using versioned directories.

> Schema modules for all Common Information MUST exist within a directory path of “/EnterpriseObjectLibrary/Core/Common/V<Y>” within the library where Y is the major version of the Core Common information.

Common information consists of DataTypes, Common Components, Meta Data, Code Lists, Web services structures, and supportive XML structures.

> The DataType schema module MUST be named “DataTypes.xsd”.
>
> The Common Components schema module MUST be named “CommonComponents.xsd”.
>
> The Meta data schema module MUST be named “Meta.xsd”.
>
> The Code Lists schema module MUST be named “CodeLists.xsd”.
>
> All Web services schema modules MUST be named according to the most generally accepted name of the file.
>
> The Supportive XML structures schema module MUST be named “XML.xsd”.

Data Types

DataTypes are used to build common components and EBOs. Data Types may consist of simple types, which correspond to the CCTS Business Data Types and Core Data Types as defined in CCTS version 2.01. The following example illustrates DataType:

```
<xsd:complexType name="IdentifierType">
  <xsd:simpleContent>
   <xsd:extension base="xsd:normalizedString">
    <xsd:attribute name="schemeID" type="xsd:normalizedString"
use="optional"/>
    <xsd:attribute name="schemeAgencyID" type="xsd:normalizedString"
use="optional"/>
    <xsd:attribute name="schemeVersionID"
type="xsd:normalizedString" use="optional"/>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
```

Common Components

Common Components are the enterprise common components that are used commonly across the EBOs. They consist of components such as Address or Charge. The following example illustrates a Common Component type:

```
<xsd:complexType name="InvoiceLineIdentificationType">
  <xsd:complexContent>
   <xsd:extension base="IdentificationType">
    <xsd:sequence>
     <xsd:element name="Description" type="TextType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
```

MetaData

Meta data is defined as Verbs, Faults, and other structures used to support EBM and integration interactions. The following example illustrates a MetaData type:

```
<xsd:complexType name="UpdateType">
  <xsd:attribute name="returnObjectCode" type="StringType"
use="optional"/>
 </xsd:complexType>
```

Code Lists

Code Lists and external Code Lists supportive information such as ascending/descending indicators is provided as common information. The following example illustrates a supportive code list information type.

```
<xsd:simpleType name="SortDirectionCodeType">
  <xsd:restriction base="xsd:normalizedString">
   <xsd:enumeration value="ASC"/>
   <xsd:enumeration value="DESC"/>
  </xsd:restriction>
 </xsd:simpleType>
```

Common Enterprise Business Objects (core)

Core Common EBOs are the most common industry-neutral business objects that are used to conduct business. These are coarse-grained objects. Core Common EBOs exist within a versioned directory. An example is BillOfMaterials. Each Core Common EBO is defined within its own namespace. For every Core Common EBO schema module, a corresponding EBO schema module exists.

Enterprise Business Objects (core)

Core EBOs are industry-neutral business objects that are used to conduct business. These are the coarse-grained objects that are traditionally used to conduct business. Core EBOs are contained within EBO-specific schema modules, such as the AccountBalanceAdjustmentEBO schema module. Each EBO is defined within its own namespace.

EBOs have associated EBM schema modules used for communication of the EBO content within the context in which a given EBO is used. EBM schema modules are specified within the same physical directory of their associated EBO. The EBM schema module is located in the EBO directory and in the same namespace as the given EBO.

Industry Content Modularity

The Industry Content modularity dimension of the Modularity Model addresses the industry-specific business information within the library. It follows, and is built on, the modularity model specified in Core.

Management Modularity

The key to supporting both Core and Industry Content modularity of the XML schema lies within management modularity of the library, which is reflected in the physical packaging of the library and based on versioning.

Versioning

Versioning is the mechanism for content evolution within the library. All content is versioned. Versioning is divided between major and minor, with specific characteristics of each.

Defining Major and Minor Versions

A major version of a schema module defines a non-backward (breakage) compatible revision from a supporting software, data structure, or semantic processing perspective. These changes consist of, but are not limited to:

- Changing element, type, and attribute names.
- Changing the structures so as to break polymorphic processing capabilities.
- Deleting or adding mandatory elements or attributes.
- Removing or changing values in enumerations.

> Every EBO Schema major release or version number with its Namespace name scheme MUST be a sequentially assigned incremental integer greater than zero.

A minor version of a schema module defines a backward-compatible (no breakage) revision from a supporting software, data structure, and semantic processing perspective. Multiple minor versions may exist within a major version.

It is imperative that as minor versions evolve for given namespace, that backward compatibility within the minor version sequence must be maintained. In the EBO XML Schema, minor versions are defined within the same namespace. Each subsequent minor version must incorporate the previous minor version revisions so as to not break backward compatibility across the minor release sequence.

These changes consist of, but are not limited to:

- Adding optional elements or attributes.
- Adding values to enumerations.

Every EBO XSD Schema minor version MUST be within scope of the namespace as the major version in which it is associated.

Minor versioning MUST be limited to changes that do not require supportive software structure and semantic processing alterations.

Minor versions of the schema MUST incorporate all XML constructs from the immediately preceding major or minor version schema.

Understanding Major and Minor Versions Mechanism

The versioning mechanism comprises two distinct approaches. In both approaches, the major version is explicitly identifiable. The two approaches are:

- Group versioning, using directory names (see Figure 1)
- Individual versioning, using indicators in individual namesspace name schemes

In both cases, however, major version numbers are always reflected within content namespace name schemes, and minor version numbers are not reflected.

Group versioning using versioned directory names is used to accumulate major revision changes for groups of related content, appropriate for evolving together over time. Multiple versions of content that are versioned using group versioning by directory name may exist within one release. This approach provides revision isolation so that changes that affect only specific groups of EBOs and the associated EBMs do not affect other groups of EBOs that do not require the changes. The directory name reflecting the major version of the group will be the same as the highest major version reflecting the namespace name scheme across the group content. Concerning minor versioning of this approach, the minor version changes are provided within the same namespace of content within the group by overwriting the previous schema module.

Group versioning within the library is used for:

- Core/Industry Common information
- Core/Industry Common EBOs
- Core/Industry Custom Common information
- Core/Industry Custom EBOs

Directory names reflecting major versions of evolving group content MUST reflect the highest major version embedded in the scheme for Namespace names across the group content.

Individual versioning consists of the major version indicator within the namespace name scheme. This approach allows revision of content on an individual basis. Concerning minor versioning of this approach, the minor version changes are provided within the same namespace of the content by simply overwriting the previous schema module.

Individual versioning within the library is used for:

- Core/Industry EBOs
- Core/Industry Custom EBOs

Chapter 5: About Context

This section discusses:

- The current operational context
- Context future

The Current Operational Context

For this document, the application of operation context means to identify the content of the EBO that is usable within a given action, or operation, and expressing only that content for the action-specific schemas. The application of context other than operation is outside the scope of this document.

Modeled business objects must be used within a specific operational context that affects the scope and applicability of the data necessary to complete the transaction. The concept of context includes more than operations, because context affects what parts of the model are used. For example, a SalesOrder EBO can be used in several business transactions, such as a "QuerySalesOrder," which is a request to send a specified order, a "DeleteSalesOrder," which tells the receiver to remove an order from active status, or a “ProcessSalesOrder,” which is a request to process the specified order. This variation of the data actually used within a business transaction for a given object is the operational context. The data needed to complete different transactions is very different. For example:

- Query needs only the information necessary for a given query.
- Delete usually needs simple identification information only.
- Process needs much more detail.

In the EBO Schema, the EBM schema modules provide the operational context for a given EBO.

Context Future

The concept of Context, beyond the current operational context, will evolve over time and will be included in the release of the EBO library.

Chapter 6: General XML/Schema

This section discusses:

- Best practices in the industry
- Namespace names and prefix
- Schema location

Best Practices in Industry

The EBO XML Schema will follow the best practices of XML Schema design available at the time this document is published. These best practices indicate that:

- All XML Schema root elements. That is, xsd:schema, should be defined using the attributes elementFormDefault=”qualified”, attributeFormDefalt=”unqualified”.
- The version=”[The version of the schema.]” attribute should be used to indicate the actual version of the schema.
- All the namespaces used within a schema must be identified in the xmlns attributes list for the schema root element of an XML Schema file.
- The targetNamespace attribute be used to identify the target namespace for the schema module.
- “xsd” be used as prefix for the XML Schema namespace.
- Upper camel case be used to name elements, types, and groups as required.
- Lower camel case be used to name attributes, as required earlier in this document.

Each schema module MUST be defined using “xsd” as the prefix of the XML Schema namespace.

All schemas MUST have one instance of the xmlns schema attribute as shown here: xmlns:xsd=”http://www.w3.org/2001/XMLSchema”

Each schema module MUST be defined with the schema attribute elementFormDefault set to “qualified,” as illustrated here: `elementFormDefault=”qualified”`.

Each schema module MUST be defined with the schema attribute attributeFormDefault set to “unqualified,” as illustrated here: `attributeFormDefault=”unqualified”`.

Each schema module MUST be defined with the schema attribute version set to the major version of the schema module, as illustrated here: version=”1.0”

All namespaces used within a schema module MUST be identified by an instance of the schema attribute xmlns.

Each schema module MUST be defined with the schema attribute targetNamespace defined where its value represents the namespace in which the schema module is defined within.

Namespace Names and Prefix

Each namespace used will have its own namespace prefix. The list of these prefixes and the corresponding namespace names is provided in the following table. Exact detail of these formats is specified throughout this document.

Preface	Namespaces
Corecomcust	http://xmlns.oracle.com/EnterpriseObjects/Core/Custom/Common/V<MajorVersionNumber>
Coreinvcust	http://xmlns.oracle.com/EnterpriseObjects/Core/Custom/EBO/Invoice/V<MajorVersionNumber>
Corecom	http://xmlns.oracle.com/EnterpriseObjects/Core/Common/V<MajorVersionNumber>
Coreinv	http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V<MajorVersionNumber>
<IndustryName>comcust	http://xmlns.oracle.com/EnterpriseObjects/Industry/<IndustryName>/Custom/Common/V<MajorVersionNumber>
<IndustryName>invcust	http://xmlns.oracle.com/EnterpriseObjects/Industry/<IndustryName>/Custom/EBO/Invoice/V0
<IndustryName>com	http://xmlns.oracle.com/EnterpriseObjects/Industry/<IndustryName>/Common/V<MajorVersionNumber>
<IndustryName>inv	http://xmlns.oracle.com/EnterpriseObjects/Industry/<IndustryName>/EBO/Invoice/V<MajorVersionNumber>

Schema Location

While namespace names within the library are in the form of a URL, consistent with W3C namespace specifications, no requirement exists for the URL value itself to be resolvable.

The location of dependent schemas within the library is specified by the use of the schemaLocation attribute, which is required across the library. To insure library consistency as a unit concerning resolvable location of schema when xsd:import and xsd:include are used, schema location values must be in relative form within the library.

All schema modules within the library that use xsd:import or xsd:include MUST use a corresponding xsd:schemaLocation attribute with a relative value within the library that identifies location of the imported or included schema module.

Chapter 7: EBO Structure

This section discusses:

- Namespace and namespace prefix
- EBO relationships

Namespace and Namespace Prefix

This section discusses:

- Namespace schemes
- Prefix tags

Namespace Schemes

EBO Schema Modules exists in namespaces which follows a namespace scheme that includes whether the EBO is within Core or Industry and its major version number.

An EBO Namespace within Core MUST use the following scheme:
http://xmlns.oracle.com/EnterpriseObjectLibrary/Core/EBO/<EBOName>/V<MajorVersion> where <EBOName> is the name of the EBO and <MajorVersion> is a positive integer reflecting the major version number of the EBO version.

An EBO Namespace within an industry MUST use the following scheme:
http://xmlns.oracle.com/EnterpriseObjectLibrary/Industry/<IndustryName>/EBO/<EBOName>/V<MajorVersion> where <IndustryName> reflects the name of the industry, <EBOName> is the name of the EBO and <MajorVersion> is a positive integer reflecting the major version number of the EBO version.

As an example, for the version zero of the Core EBO Invoice, the namespace for this schema model is “http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Invoice/V0”.

As an example, if the EBO for the Telco industry is named Invoice, the namespace for this schema models is http://xmlns.oracle.com/EnterpriseObjects/Industry/Telco/EBO/Invoice/V0

Prefix Tags

A consistent approach is defined for EBO namespace prefixes.

In the case of Core EBOs, the leading fixed tag is “core,” and it is followed by a unique abbreviation of the EBO name.

In the case of Industry EBOs, the leading fixed tag is the name of the industry, followed by a set of unique abbreviations qualifying the type of EBO (such as “com” for common or “cust” for custom), followed by a unique abbreviation of the EBO name.

The namespace prefix in all of the schema modules for the Core EBOs MUST be of the form: core<eboname> where the <eboname> is a unique abbreviation of the EBO name, all lower case.

The namespace prefix in schema modules for Industry EBOs MUST be of the form: <industryName><QualifyingAbbreviations><EBOName> where the <industryName> is the name of the industry, <QualifyingAbbreviations> is a set of tags indicating the type of EBO and <EBOName> is a unique abbreviation of the EBO name, all lower case.

As an example, for the Core EBO Invoice, the namespace prefix is “coreinv”.

An example of a Telco Custom Common EBO for the telco industry would be “telcocomcust”.

EBO Relationships

This section discusses:

- EBO naming
- Relationships

EBO Naming

EBO naming is based on UN/CEFACT Core Components naming rules and uses global xsd:elements typed as xsd:complexTypes using EBO-specific naming conventions.

Upper camel case (UCC) MUST be used to name all EBOs xsd:element and xsd:complexType.

All EBOs MUST be represented as a global xsd:element using the name of the EBO ending in the word “EBO” and typed as an xsd:complexType using the name of the EBO ending in the word “EBOType”.

An example is:

```
<xsd:element name="AccountBalanceAdjustmentEBO"
type="AccountBalanceAdjustmentEBOType">
```

Relationships

Relationships between EBOs and EBOs, data types, or common components can take one of two forms: either a whole/part relationship or a relationship based on identification.

Whole/Part Relationship

Whole/part relationships define a relationship in which the part is required to complete the whole as an independent unit, and the whole and the part coexist within the same life cycle. This is sometimes known as a ”by-value,” and also referred to as a Whole/Part relationship.

An EBO can represent the whole and have part relationships. This relationship is specified by the use of a local element within the whole, which is typed as a global type defining its structure. Cardinality is also specified.

Whole/Part relationships defined within an EBO MUST use a local xsd:element typed as a global type that defines the structure of the part.

Whole/Part relationships exist between and EBOs and data types within the DataTypes schema module.

A relationship between an EBO and a DataType MUST use a local xsd:element typed as a global DataType type in the DataTypes schema module.

An example is:

```
<xsd:element name="Name" type="corecom:NameType" minOccurs="0">
    <xsd:annotation>
     <xsd:documentation>
Name of the Pricelist
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>
```

Identification Relationship

Identification relationships define a relationship in which an independent life cycle exists between the involved entities. This is sometimes known as a ”by-reference” relationship. Within the library, this is accomplished by the use of identification, and is referred to as an Identification Relationship.

This is done using reference objects within the CommonComponents schema module. Reference objects can be specified with various structures, but ultimately must use the IdentificationType in the CommonComponents schema module. Describing the IdentificationType and its semantics is beyond the scope of this specification.

EBOs MUST use an xsd:ref attribute referring to a reference component within the CommonComponents schema module for Identity Relationships.

Reference components within the CommonComponents schema module MUST ultimately use the global IdentificationType within the CommonComponents schema module.

The following partial structures are an example of an Identification Relationship:

A global complex type within an EBO schema module uses an xsd:ref to refer to a global reference element within the CommonComponents schema module:

```
<xsd:complexType name="PriceListSubscriberType">
  <xsd:sequence>
………
   <xsd:element ref="corecom:SubscriberPartyReference"
minOccurs="0">
……..
 </xsd:complexType>
```

The global reference element referred to within the CommonComponents schema module: (It is typed as a global xsd:complexType)

```
<xsd:element name="SubscriberPartyReference"
type="SubscriberPartyReferenceType">
  <xsd:annotation>
   <xsd:documentation>Person or Organization that subscribes to a
service. Examples are catalog subscriptions, broadcasting service
subscriptions, cellular phone service subscriptions
etc</xsd:documentation>
  </xsd:annotation>
 </xsd:element>
```

In this case, the type of the reference element is an extension within the CommonComponents schema module:

```
<xsd:complexType name="SubscriberPartyReferenceType">
  <xsd:complexContent>
   <xsd:extension base="PartyReferenceType">
…..
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
```

The base type includes an xsd:ref referring to an identification global element within the CommonComponents schema module:

```
<xsd:complexType name="PartyReferenceType" abstract="true">
  <xsd:sequence>
   <xsd:element ref="PartyIdentification" minOccurs="0"/>
…………
  </xsd:sequence>
 </xsd:complexType>
```

The global PartyIdentification element within the CommonComponents schema module (typed as an xsd:complexType):

```
<xsd:element name="PartyIdentification"
type="PartyIdentificationType">
  <xsd:annotation>
   <xsd:documentation>Unique Identification of a
Party</xsd:documentation>
  </xsd:annotation>
 </xsd:element>
```

In this case, the type of the PartyIdentification global element is an extension of the IdentificationType within the CommonComponents schema module:

```
<xsd:complexType name="PartyIdentificationType">
  <xsd:complexContent>
   <xsd:extension base="IdentificationType"/>
  </xsd:complexContent>
 </xsd:complexType>
 <xsd:element name="PartyIdentification"
type="PartyIdentificationType">
  <xsd:annotation>
   <xsd:documentation>Unique Identification of a
Party</xsd:documentation>
  </xsd:annotation>
 </xsd:element>
```

The base IdentificationType within the CommonComponents schema module specifies the ability to identify components:

```
<xsd:complexType name="IdentificationType">
  <xsd:sequence>
   <xsd:element name="BusinessComponentID" type="IdentifierType"
minOccurs="0">
    <xsd:annotation>
     <xsd:documentation>Unique Key for the application agnostic
representation of the objectinstance. Business Documents generated
by AIA applications will have the BusinessComponentID necessarily
populated.
The BusinessComponentID will be generated using the API provided by
AIA Infrastructure</xsd:documentation>
    </xsd:annotation>
   </xsd:element>
…………………………….
```

```
     </xsd:sequence>
    </xsd:complexType>
```

EBO Customization

A local element named “Custom” exists at the end of every EBO type. The Custom element is defined to use a type named after the type it is added to, with the word Custom prepended. Additionally, this custom type is defined in the Core Custom common version of the common files namespace and directory structure.

Each Common Components complexType MUST have a local element added named “Custom”.

The Custom element exists at the end of each Common Components complexType MUST use a type whose name is in the form: Custom<complexTypeName that is having Custom element added>. This Custom type is defined in the Core Custom Common in the corresponding version namespace and directory.

Chapter 8: EBM Structure

This section discusses:

- Operation context, interchange, and naming conventions
- Operation-specific request/response types
- Operation-specific request/response data area types
- Operation-specific request/response EBM types

Operational Context, Interchange, and Naming Conventions

One of the key advantages of the EBO Schema is that it is based upon the OAGi OAGIS Business Object Document (BOD) structure for messaging.

EBOs within Core or an Industry have an operational context applied manifested in the Enterprise Business Messages (EBMs). Operational context essentially means a verb, which coincides with a specific message structure for purposes of interchanging EBO data. One EBM schema module exists per EBO. It is in the same directory in the library as its EBO, within the same namespace, and includes the EBO content.

One EBM schema module MUST exist per EBO.

The location of an EBM schema module MUST be in the same directory as its corresponding EBO.

An EBM schema module must be defined within the same namespace as its corresponding EBO.

An EBM schema module must use xsd:include to include its corresponding EBO.

Operation patterns exist within EBM schema modules and are defined according to business use, subject matter experts, and requirements.

The concept of a “group-operation-pattern” is used when defining EBOs. This is an informal term used to refer to generally accepted groups of operations to support a given interaction style, such as with CRUD (Create/Read(query)/Update/Delete), which has its genesis in database operations, Pub/Sub (topic publication and subscribe) operations for notification systems, and so on.

A typical CRUD group-operation-pattern exists for EBOs within many of the EBM schema modules, with some additional operations such as process, sync, and so on. Other group-operation-patterns may emerge over time. Regardless of the exact group-operation-pattern appropriate for specific EBOs, a set of structures is always consistently defined within and EBM schema module based on a Request/Response message exchange pattern following a set of naming conventions.

Including EBO Content

Including the EBO content in the EBM schema module permits usage of the EBO data within messages, and is accomplished by extending the EBO type. The name of the module is a combination of the EBO name plus “EBMType”. This type is used for messages requiring all potential content defined in the EBO definition (further explanation follows).

> An EBM schema module MUST define an xsd:complexType as an extension of the xsd:include(d) EBO Type, with a name of “<EBOname>EBMType”, where EBOname is the name of the EBO if any messages require the full EBO content.

An example is:

```
<xsd:complexType name="CustomerPartyEBMType">
  <xsd:complexContent>
   <xsd:extension base="CustomerPartyEBOType"/>
  </xsd:complexContent>
 </xsd:complexType>
```

Operation-Specific Request/Response Types

Each operation that is appropriate for a given EBO requires a request type and a response type.

The naming convention for an EBO request type for a given operation is a combination of the operation, EBO name, and “Type”, while the naming convention for the response type is the operation, EBO name, and “ResponseType”. Global elements are also defined and are typed as such.

> For every EBO operation, an xsd:complexType MUST be defined in the form <Operation><EBOName>Type, where <Operation> is the name of the operation and <EBOName> is the name of the EBO.
>
> For every EBO operation, an xsd:complexType MUST be defined in the form <Operation><EBOName>ResponseType, where <Operation> is the name of the operation and <EBOName> is the name of the EBO.
>
> For every EBO operation, a global xsd:element MUST be defined in the form <Operation><EBOName>, where <Operation> is the name of the operation and <EBOName> is the name of the EBO, and MUST be typed as its corresponding xsd:complexType.
>
> For every EBO operation, a global xsd:element MUST be defined in the form <Operation><EBOName>Response, where <Operation> is the name of the operation and <EBOName> is the name of the EBO, and MUST be typed as its corresponding xsd:complexType.

Examples are:

```
<xsd:complexType name="CreateCustomerPartyType">
…..
<xsd:element name="CreateCustomerParty"
type="CreateCustomerPartyType"/>
…..
<xsd:complexType name="CreateCustomerPartyResponseType">
….
<xsd:element name="CreateCustomerPartyResponse"
type="CreateCustomerPartyResponseType"/>
…
```

The content defined within an operation's request type and response type varies depending on the intention of the operation. Some types may require the full EBO content and some may not. Specifying the full EBO content is facilitated by extending the defined EBO EBM type.

EBM Request Types and Response Types that specify the entire EBO content MUST extend the extended EBO EBM type within their EBM schema module.

Here is an example of specifying full EBO content for an operation's request type:

```
<xsd:complexType name="CreateCustomerPartyType">
  <xsd:complexContent>
   <xsd:extension base="CustomerPartyEBMType"/>
  </xsd:complexContent>
 </xsd:complexType>
```

Specifying alternate content within an operation's request type or response type can take many forms appropriate to the operation, request/response, and EBO. Here is an example of specifying an identifier by leveraging the Identification structure in Core for an operation's request type:

```
<xsd:complexType name="DeleteCustomerPartyType">
  <xsd:sequence>
   <xsd:element ref="corecom:Identification" minOccurs="0"/>
   <xsd:element name="Custom"
type="corecustomerpartycust:CustomCustomerPartyEBOType"
minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>
```

Operation-Specific Request/Response Data Area Types

Each operation that is appropriate for a given EBO requires a request data area type and a response data area type.

The naming convention for an operation's data area type for a request is a combination of the operation, EBO name, and "DataAreaType", while the naming convention for an operation's data area for a response is the operation, EBO name, and "ResponseDataAreaType".

For every EBO operation, an xsd:complexType MUST be defined in the form <Operation><EBOName>DataAreaType, where <Operation> is the name of the operation and <EBOName> is the name of the EBO.

For every EBO operation, an xsd:complexType MUST be defined in the form <Operation><EBOName>ResponseDataAreaType, where <Operation> is the name of the operation and <EBOName> is the name of the EBO.

The content defined within the operation's request data area type and response data area type includes an xsd:ref reference to the operation's appropriate request type or response type, which defines the specific content for the data area (and is sometimes called the noun area).

Along with the reference to the data structure, an operation's request data type area contains operational context (or verb) MetaData leveraged from the Core Meta schema module, while an operation's response data area type contains matching operational context response MetaData from the Core Meta schema module. These Meta structures provide the ability carry additional, operation-specific information that is required.

All EBM Request DataAreaTypes MUST specify an xsd:element using xsd:ref referencing the operation's global xsd:element RequestType.

All EBM Response DataAreaTypes MUST specify an xsd:element using xsd:ref referencing the operation's global xsd:element ResponseType.

All EBM Request DataAreaTypes MUST specify an xsd:element using xsd:ref referencing the appropriate global xsd:element operational context (verb) form the Core Meta schema module.

All EBM Response DataAreaTypes MUST specify an xsd:element using xsd:ref referencing the appropriate global xsd:element operational context (verb) response from the Core Meta schema module.

Here is an example of an operation's request data area type:

```
<xsd:complexType name="CreateCustomerPartyDataAreaType">
  <xsd:sequence>
   <xsd:element ref="corecom:Create"/>
   <xsd:element ref="CreateCustomerParty"/>
  </xsd:sequence>
 </xsd:complexType>
```

Here is an example of an operation's response data area type:

```
<xsd:complexType name="CreateCustomerPartyResponseDataAreaType">
  <xsd:sequence>
   <xsd:element ref="corecom:CreateResponse"/>
   <xsd:element ref="CreateCustomerPartyResponse"/>
  </xsd:sequence>
 </xsd:complexType>
```

Operation-Specific Request/Response EBM Types

Each operation that is appropriate for the EBO requires a request EBM type and a response EBM type.

The naming convention for an operation's request EBM type is a combination of the operation, EBO name, and "EBMType", while the naming convention for an operation's response EBM type is the operation, EBO name, and "ResponseEBMType". Global elements using similar name conventions also exist, which are typed to the corresponding type.

For every EBO operation, an xsd:complexType MUST be defined in the form <Operation><EBOName>EBMType, where <Operation> is the name of the operation and <EBOName> is the name of the EBO.

For every EBO operation, an xsd:complexType MUST be defined in the form <Operation><EBOName>ResponseEBMType, where <Operation> is the name of the operation and <EBOName> is the name of the EBO.

For every EBO operation, a global xsd:element MUST be defined in the form <Operation><EBOName>EBM, where <Operation> is the name of the operation and <EBOName> is the name of the EBO, and MUST be typed as its corresponding xsd:complexType.

For every EBO operation, an xsd:complexType MUST be defined in the form <Operation><EBOName>ResponseEBM, where <Operation> is the name of the operation and <EBOName> is the name of the EBO, and MUST be typed as its corresponding xsd:complexType.

Examples are:

```
<xsd:complexType name="CreateCustomerPartyEBMType">
…..
<xsd:element name="CreateCustomerPartyEBM"
type="CreateCustomerPartyEBMType"/>
```

```
…..
<xsd:complexType name="CreateCustomerPartyResponseEBMType">
….
<xsd:element name="CreateCustomerPartyResponseEBM"
type="CreateCustomerPartyResponseEBMType"/>
…
```

The content defined within both the operation's request EBM type and response EBM type includes a languageCode attribute typed as LanguageCodeType from the Core Common CodeLists schema module and a versionID attribute typed as a StringType from the Core common DataTypes schema module.

> For every operation's Request EBM Type and Response EBM Type, there MUST exist an attribute named "languageCode" typed as LanguageCodeType from the Core Common CodeLists schema module.
>
> For every operation's Request EBM Type and Response EBM Type, there MUST exist an attribute named "versionID" typed as StringType from the Core Common DataTypes schema module.

The content defined within both the operation's request EBM type and response EBM type also includes a local element named DataArea, which is typed as the operation's appropriate request or response data type.

> For every operation's Request EBM Type, there MUST exist a local xsd:element named "DataArea," which is typed as the operation's Request Data Type.
>
> For every operation's Response EBM Type, there MUST exist a local xsd:element named "DataArea," which is typed as the operation's Response Data Type.

The content defined within both the operation's request EBM Type and response EBM Type also includes a local element reference to the global EBMHeader element within the Core Common Meta schema module.

The EBMHeader communicates information about the sending application that the receiving application needs to know to receive the message and further understand what it is to do with the message. This information includes whether the sending application requires a response to know that the message was received and understood, whether it received and had errors, or that the sending application does not need to know that the message was received at all, as is the case in a fire and forget scenario.

When receiving a message, the receiving application must verify that the sending application is a trusted source, has the authority to request an action to be performed, or both. The EBMHeader communicates the authorization codes of the sending application and the user of the sending application. The receiving application is responsible for determining whether these authorizations are sufficient to perform the given action.

Each message that is communicated must be able to be uniquely identified. The EBMHeader provides a place in which to communicate this globally unique identifier (GUID). The sending application or its proxy is responsible for providing this GUID. This is often helpful in logging, tracking, and auditing information as it is communicated across enterprises.

> For every operation's Request EBM Type and Response EBM Type, there MUST exist a local xsd:element using the xsd:ref attribute to refer to the global xsd:element EBMHeader in the Core Common Meta schema module.

Here is an example of an operation's response EBM type:

```
<xsd:complexType name="CreateCustomerPartyResponseEBMType">
  <xsd:sequence>
```

```
      <xsd:element ref="corecom:EBMHeader"/>
      <xsd:element name="DataArea"
   type="CreateCustomerPartyResponseDataAreaType"/>
     </xsd:sequence>
     <xsd:attribute name="languageCode"
   type="corecom:LanguageCodeType"/>
     <xsd:attribute name="versionID" type="corecom:StringType"/>
    </xsd:complexType>
```

Chapter 9: References

[Keywords] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[URI] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998, http://www.ietf.org/rfc/rfc2396.txt

[XML] Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/REC-xml

[XMLSchema1] W3C Recommendation, XML Schema Part 1: Structures, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[XMLSchema2] W3C Recommendation, XML Schema Part 2: Datatypes, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

[OAGINDR] OAGi OAGIS 9.0 Naming and Design Rules Standard version 0.7, Published September 30, 2005

[CCTS] UN/CEFACT Core Components Technical Specification Draft 2.2, Published March 31, 2006

[UNCEFACTNDR] UN/CEFACT XML Naming and Design Rules, Draft 2.0, Published January 20, 2006 http://www.unece.org/cefact/xml/xml_index.htm

Index