Semantic Refinement Tool:

OAGIS Repository Release 1.0 Overview

Editor: Serm Kulvatunyou, NIST

Creation Date: 08/11/2016

Last Updated: 9/22/2016 7:34 AM

Version: 1.0

Status: IN PROCESS

Contributors:

Hakju Oh, NIST

Taehun Kim, NIST

Jeff Rice, Boeing

Michael Rowell, Oracle

David Connelly, OAG

Scott Nieman, Land O’Lakes

Steffen Fohn, ADP

Change Tracking

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Detail |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[3. Introduction 4](#_Toc461728550)

[3.1 Purpose 4](#_Toc461728551)

[3.2 Scope 4](#_Toc461728552)

[3.3 Definitions, Acronyms and Abbreviations 4](#_Toc461728553)

[3.4 References 5](#_Toc461728554)

[4. Content Stored as BLOB 6](#_Toc461728555)

[5. Content imported as individual database entity 7](#_Toc461728556)

[5.1 xsd:simpleType with no xsd:enumeration and no xsd:pattern (BDT) 7](#_Toc461728557)

[5.2 xsd:simpleType with no xsd:enumeration but with xsd:pattern 8](#_Toc461728558)

[5.3 xsd:simpleType with xsd:enumeration 8](#_Toc461728559)

[5.3.1 Code List 8](#_Toc461728560)

[5.3.2 Agency ID List 9](#_Toc461728561)

[5.4 xsd:complexType with xsd:simpleContent (BDT) 10](#_Toc461728562)

[5.5 xsd:complexType with complex content (ACC) or xsd:group definitions (ACC & ASCCP) 11](#_Toc461728563)

[5.6 Global xsd:element (BCCP) or xsd:attribute declaration using a BDT (BCCP & BCC) 13](#_Toc461728564)

[5.7 Global xsd:element declaration using the ACC (ASCCP) 15](#_Toc461728565)

[5.8 xsd:element or xsd:group reference (ASCC & BCC) 15](#_Toc461728566)

[5.9 Local xsd:element with global type (ASCCP & ASCC) 16](#_Toc461728567)

[5.10 xsd:any 17](#_Toc461728568)

[5.11 Documentation 18](#_Toc461728569)

[5.12 How the sequencing key is assigned? 18](#_Toc461728570)

[6. Mapping Summary 19](#_Toc461728571)

# Introduction

## Purpose

The purpose of this document is to give an overview of how OAGIS 10.x Model content is stored in the OAGIS repository. The purpose is not to describe how all kinds of artifacts in the Model schema are mapped. It will particularly focus on how the name of each OAGIS type and element appear in the database, because search and query for components by names and definitions are the target use cases of the repository.

## Scope

OAGIS repository database schema is based on the UN/CEFACT Core Component Specification. Therefore, this document assumes the reader has familiarity with Core Component Specification (CCS, or formerly known as Core Component Technical Specification – CCTS) and the Core Data Type Catalog (CDTC). The repository is based on the CCS 3.0 and CDTC 3.1.

Although the database schema includes schemes for managing the Business Information Entities (BIEs), this document only covers the Core Component (CC) part of the schema.

The CC part of the database schema includes columns to enable multiple releases of OAGIS Model to be stored in a normalized form (i.e., to keep only the delta between releases). However, the current version of the repository does not implement that capability yet. In other words, the repository stores only the latest release of the OAGIS model.

## Definitions, Acronyms and Abbreviations

CCS = Core Component Specification

CCTS = Core Component Technical Specification

CCS = CCTS (CCS is a new name of the CCTS)

CDTC = Core Data Type Catalog

BIE = Business Information Entity.

CC = Core Component

ACC = Aggregate Core Component

ASCC = Association Core Component

ASCCP = Association Core Component Property

BCC = Basic Core Component

BCCP = Basic Core Component Property

DT = Data Type. It covers both CDT and BDT.

CDT = Core Data Type

BDT = Business Data Type

DT = Data Type. This refers to both BDT and CDT.

Default BDT = A BDT that has UUID in the name, e.g., TextType\_0VCBZ5

Unqualified BDT = A BDT that is directly derived from a default BDT

SC = Supplementary Component

DEN = Dictionary Entry Name

OAGIS Model or OAGIS Model Content = OAGIS canonical XML Schemas residing the Model folder of the OAGIS Enterprise Edition.

OAGSRT WG = The OAGi Semantic Refinement Method and Tool Working Group, which produced this document.

Value domain restriction = This string of relationships between tables that convey the assignment of a data type of a particular expression (e.g., xsd:decimal), a code list, or an agency ID list to a DT or a DT SC.

Built-in type = a data type of a specific expression. Built-in type includes not only types defined by the specification of an expression, but also those created by OAGIS that are variations of the CDTC’s primitives (i.e., types defined in XMLSchemaBuiltintype\_1.xsd and XMLSchemaBuiltinType\_1\_patterns.xsd).

## Convention

Database table and column names are in all upper cases and each word is separated by the under bar character using this body font such as DT\_ID.

A data value is surrounded by a single quote, except when it is a number. It is in the same body font as the database table and column names, e.g., DEN = ‘Party. Details’.

Generally, this same body font is used for system values, XML schema entities, XML schema type and element names, etc. that are in-line with the text.

## References

CCS ….

CDTC ….

Data model file …

# Content Stored as BLOB

The following schemas are stored as BLOB in the repository.

* All files in the folder Model\Platform\2\_1\Common\DataTypes
* All files in the folder Model\Platform\2\_1\Common\ISO20022
* Files whose names ending with ‘IST’ in the Model\BODs and Model\Nouns folders
* Model\Platform\2\_1\OAGi-Platform.xsd

It should be noted that some content in the Model\Platform\2\_1\Common\DataTypes folder is imported. In particular, types defined in the BusinessDataType\_1.xsd that are used by other schemas are imported. For example, the CodeType\_1E7368 type is imported because it is used in schemas such as Fields.xsd.

There is a plan to import all content in the Model\Platform\2\_1\Common\DataTypes folder in the next version of the repository.

The Model\Platform\2\_1\OAGi-Platform.xsd has no component definition but only three xsd:include statements.

Content in the Model\Platform\2\_1\Common\ISO20022 folder and those schemas whose names ending with ‘IST’ is not imported because it is a third party content that does not follow OAGIS schema architecture.

# Content imported as individual database entity

## xsd:simpleType with no xsd:enumeration and no xsd:pattern (BDT)

This kind of type definitions is found in the Fields.xsd and BusinessDataType\_1.xsd.

Such type definition is considered a BDT and is imported into DT as a main table. It may or may not have associated entry in the DT\_SC table, which represents the supplementary component (SC) of a BDT. If the type definition is based on another BDT that is theoretically based on a CDT having some SCs, then it will have some inherited DT\_SC entries, however the CARDINALITY\_MAX column of the DT\_SC will be zero.

In other words, SCs are inherited by copy. Therefore, copies of all SCs are created per the based DT. The SCs that are not used in the derived DT have zero CARDINALITY\_MAX.

**Example 1**

Below is the schema definition (with the annotation removed) of the TextType\_0VCBZ5 from the BusinessDataType\_1.xsd. It is imported into the DT and DT\_SC tables as follows:

<xsd:simpleType name="TextType\_0VCBZ5" id="oagis-id-42a03ed19450453da6c87fe8eadabfa4">

<xsd:restriction base="xsd:normalizedString"/>

</xsd:simpleType>

A record is created in the DT table with DEN = ‘Text\_0723C8. Type’ and DATA\_TYPE\_TERM = ‘Text’. The fact that the type is based on the xsd:normalizedString is captured by a string of relationships from BDT\_PRI\_RESTRI table to the CDT\_AWD\_PRI\_XPS\_TYPE\_MAP that eventually leads to the XBT table. The XBT table stores the XML schema built-in type such as the xsd:normalizedString. This string of relationships that allows for the assignment of a data type of a particular expression, a code list, or an agency ID list to a DT is described later. We’ll call this kind of relationships, ***value domain restriction***.

The created DT record will have its BASED\_DT\_ID column pointing to the Text CDT. In fact, all imported DTs are either based on a CDT or another BDT. The ‘Text. Type’ CDT is not defined in the OAGIS schema (it is not possible), but it is represented in the repository according to the CDTC specification.

In addition, a record is created in the DT\_SC table with the PROPERTY\_TERM = ‘Language’, REPRESENTATION\_TERM = ‘Code’; however, with CARDINALITY\_MAX = 0. This is because the based DT (the ‘Text. Type’ CDT) has the ‘Language Code’ supplementary component. The record also has the BASED\_DT\_SC\_ID column pointing to the ‘Language Code’ SC of the ‘Text. Type’ CDT. Therefore, it also inherits other information related to the SC, such as the value domain restriction, from the based SC. In fact, only an SC that does not exist in the CDTC will not have a based SC.

It should be noted that the BDT that has UUID in the name such as the above text type is called ***default BDT*** in the OAGSRT WG lingo.

**Example 2**

The NormalizedStringType in Fields.xsd with the below XML schema definition is imported as follows:

<xsd:simpleType name="NormalizedStringType" id="oagis-id-d26b22f9103744edb0a4d3728aefc26e">

<xsd:restriction base="TextType\_0VCBZ5"/>

</xsd:simpleType>

A record is created in the DT table with DEN = ‘Normalized String. Type’ and DATA\_TYPE\_TERM = ‘Text’. In fact, the data type term is always the same for BDTs that are derived from the same CDT. The BASED\_DT\_ID column of the DT record points to the DT record in the Example 1.

A DT\_SC record similar to those in Example 1 above is also created for the inherited ‘Language Code’ SC but again with the CARDINALITY\_MAX = 0. Its BASED\_DT\_SC\_ID column will point to the DT\_SC record created in Example 1.

It should be noted that the BDT such as this one, which is directly derived from the default BDT, is called ***unqualified BDT*** in the OAGSRT WG lingo.

## xsd:simpleType with no xsd:enumeration but with xsd:pattern

This kind of type definitions is found in the XMLSchemaBuiltinType\_1.xsd and XMLSchemaBuiltinType\_1\_patterns.xsd. At present, such type definition is not imported as individual database entity but as BLOB, except the xbt\_BooleanTrueFalseType. This type is imported into the XBT table.

## xsd:simpleType with xsd:enumeration

### Code List

This kind of type definitions is found in code list schemas in the CodeLists folder such as CodeLists\_1.xsd. Type definitions in those schemas are imported into CODE\_LIST as a main table.

Typically, a code list is defined with a pair of type definitions (i.e., two types but one entry in the CODE\_LIST table). First, a code list enumeration type is defined with enumerated values (although some don’t have any enumerated value) and then another type is defined on top of the first type. The first type usually has a naming pattern XyzCodeEnumerationType and then the latter has a naming pattern XyzCodeContentType. In addition, the XyzCodeContentType is typically a union of xsd:token and the XyzCodeEnumerationType.

If the XyzCodeContentType is a union of xsd:token and another AbcCodeContentType, the latter is then the base code list of the former (hence the BASED\_CODE\_LIST\_ID column is not NULL). In the Repository, code values are inherited by copy.

There are a few XyzCodeContentType that has no corresponding XyzCodeEnumerationType. In such case, the ENUM\_TYPE\_GUID column has a NULL value.

It is important to note that there is another set of types with same naming pattern, XyzCodeContentType, defined in the Fields.xsd. These are imported as BDTs per section 5.1 as they are constructed with the xsd:simpleType that wraps around the code list xsd:simpleType described in this section. These BDTs are typically used with a BCCP that is manifested as an xsd:attribute, because they don’t have any SC.

**Example 1**

Below is the schema snippet for a code list from the CodeLists\_1.xsd. The oacl\_ActionCodeContentType is imported into the CODE\_LIST table with NAME = ‘oacl\_ActionCode’. The GUID of the oacl\_ActionCodeEnumerationType is simply assigned to the ENUM\_TYPE\_GUID column of the created CODE\_LIST record.

All the enumeration values are created as records in the CODE\_LIST\_VALUE table.

<xsd:simpleType name="oacl\_ActionCodeContentType" id="oagis-id-49c1788460864e99b0872d0a6e58bddb">

<xsd:union memberTypes="oacl\_ActionCodeEnumerationType xsd:token"/>

</xsd:simpleType>

<xsd:simpleType name="oacl\_AgencyRoleCodeEnumerationType" id="oagis-id-1cfb7a2e0cc04586a6732f3871d670ec">

<xsd:restriction base="xsd:token">

<xsd:enumeration value="Customer"/>

<xsd:enumeration value="Supplier"/>

<xsd:enumeration value="Manufacturer"/>

<xsd:enumeration value="Broker"/>

<xsd:enumeration value="Carrier"/>

</xsd:restriction>

</xsd:simpleType>

**Example 2**

Below is a code list definition from the CodeLists\_1.xsd. The oacl\_MIMECodeContentType is a union of another code content type and the xsd:token (instead of an oacl\_MIMECodeEnumerationType), so a corresponding CODE\_LIST record will have its BASED\_CODE\_LIST\_ID column pointing to the code list corresponding to the other union member, clmIANAMIMEMediaType20090304\_MIMEMediaTypeCodeContentType.

<xsd:simpleType name="oacl\_MIMECodeContentType" id="oagis-id-51ed08216606417b9b9266b464c204f7">

<xsd:union memberTypes="clmIANAMIMEMediaType20090304\_MIMEMediaTypeCodeContentType xsd:token"/>

</xsd:simpleType>

### Agency ID List

This kind of type definitions is found in the IdentifierScheme folder. Types definition in that folder are imported into AGENCY\_ID\_LIST as a main table.

Similar to the case of the code list, an agency identification list is defined with a pair of type definitions – an XyzAgencyIdentificationContentType and an XyzAgencyIdentificationContentEnumerationType. The two types correspond to only one entry in the AGENCY\_ID\_LIST table.

**Example 3**

Below is a snippet of the agency ID list definition in the IdentifierScheme\_AgencyIdentification\_3055\_D08B.xsd. A record in the AGENCY\_ID\_LIST table is created for the clm63055D08B\_AgencyIdentificationContentType. The id attribute of the clm63055D08B\_AgencyIdentificationContentEnumerationType is simply assigned to the ENUM\_TYPE\_GUID column of the created AGENCY\_ID\_LIST record. The NAME column is hardcoded to ‘Agency Identification’. All the enumeration values are imported into the AGENCY\_ID\_LIST\_VALUE table.

<xsd:simpleType name="clm63055D08B\_AgencyIdentificationContentType" id="oagis-id-f1df540ef0db48318f3a423b3057955f">

<xsd:union memberTypes="xsd:token clm63055D08B\_AgencyIdentificationContentEnumerationType"/>

</xsd:simpleType>

<xsd:simpleType name="clm63055D08B\_AgencyIdentificationContentEnumerationType" id="oagis-id-68a3c03a4ea84562bd783fe2dc8f5487">

<xsd:restriction base="xsd:token">

<xsd:enumeration value="1"/>

<xsd:enumeration value="3"/>

<xsd:enumeration value="4"/>

…….

</xsd:simpleType>

Note that both CODE\_LIST and AGENCY\_ID\_LIST tables have a foreign key column AGENCY\_ID\_LIST\_VALUE\_ID that points to the AGENCY\_ID\_LIST\_VALUE table. The column uses the value from this import to represent the organization that maintain the code list and the agency ID list.

## xsd:complexType with xsd:simpleContent (BDT)

Similar to section 5.1, this kind of type definitions is imported as a BDT in the DT table. Any xsd:attribute presented within the type definition results in an entry in the DT\_SC table. The general approach regarding inheritance of the SC described in 5.1 also applies here.

Most of these type definitions reside in the Fields.xsd and BusinessDataType\_1.xsd. There are a few of these in Meta.xsd (e.g., ExpressionType).

**Example 1**

Below is the schema definition without annotation of the AmountType\_0723C8 from the BusinessDataType\_1.xsd. It is imported into the DT and DT\_SC tables as follows:

<xsd:complexType name="AmountType\_0723C8" id="oagis-id-e6f93bd0dc934ab2af11bd46888c1233">

<xsd:simpleContent>

<xsd:extension base="xsd:decimal">

<xsd:attribute name="currencyCode" type="oacl\_CurrencyCodeContentType" use="optional" id="oagis-id-0dd62519460d4a91bfdbc1f7778befac">

</xsd:attribute>

</xsd:extension>

</xsd:simpleContent>

</xsd:complexType>

A record is created in the DT table with DEN = ‘Amount\_0723C8. Type’ and DATA\_TYPE\_TERM = ‘Amount’. The fact that the type is based on the xsd:decimal is captured via the value domain restriction described in section 5.11.

The created DT record will have its BASED\_DT\_ID column pointing to the ‘Amount’ CDT.

In addition, a record is created in the DT\_SC table with the PROPERTY\_TERM = ‘Currency’, REPRESENTATION\_TERM = ‘Code’, CARDINALITY\_MAX = 1, and CARDINALITY\_MIN = 0. It also has the BASED\_DT\_SC\_ID column pointing to the ‘Currency’ DT\_SC of the ‘Amount. Type’ CDT. The fact that the currency code attribute is tied to a code list is captured via the value domain restriction described in section 5.11.

The REPRESENTATION\_TERM of the SC is generally the same as that of the CDT on which the BDT is based, e.g., “Code” for the “Currency” SC of BDTs based on the ‘Amount. Type’ CDT. For a new SC that does not exist in the CDTC, it is the last camel case token of the name, but if the last token is not one of the valid CDTC representation terms then the representation term is ‘Text’. E.g., if the name is ‘sequenceNumber’, the REPRESENTATION\_TERM is ‘Number’, if the name is “expressionLanguage”, the REPRESENTATION\_TERM is “Text”.

**Example 2**

The AmountType in Fields.xsd with the below XML schema definition is imported as follows:

<xsd:complexType name="AmountType" id="oagis-id-109055a967bd4cf19ee3320755b01f8d">

<xsd:simpleContent>

<xsd:extension base="AmountType\_0723C8"/>

</xsd:simpleContent>

</xsd:complexType>

A record is created in the DT table with DEN = ‘Amount. Type’ and DATA\_TYPE\_TERM = ‘Amount’. The DT record will have its BASED\_DT\_ID column pointed to the DT record in Example 1.

A DT\_SC record similar to that in Example 1 is also created for the inherited ‘Currency’ supplementary component. Its BASED\_DT\_SC\_ID column will point to the DT\_SC record created in Example 1. It should be noted that the GUID of the DT\_SC is newly generated, since it is not available in the schema. This means that each SC is treated as intrinsic to each data type – even though it is inherited and has the same semantic name.

**Example 3**

The OpenAmountType definition is shown below. It is considered a *qualified BDT* in the OAGSRT WG lingo and is imported as follows.

<xsd:complexType name="OpenAmountType" id="oagis-id-4b47b06600344fdb95bbe52f664c5a20">

<xsd:simpleContent>

<xsd:extension base="AmountType">

<xsd:attribute name="typeCode" type="xsd:token" id="oagis-id-63d46f46e4fa46ec8cc269212d7cfa77"/>

</xsd:extension>

</xsd:simpleContent>

</xsd:complexType>

A record is created in the DT table with DEN = ‘Open\_ Amount. Type’ and DATA\_TYPE\_TERM = ‘Amount’. The DT record will have its BASED\_DT\_ID column pointed to the DT record in Example 2.

A DT\_SC record for the currencyCode attribute is created along with the value domain restriction based on the inheritance from the DT\_SC record in Example 2. In addition, a DT\_SC record for the typeCode attribute with the PROPERTY\_TERM = ‘Type’ and REPRESENTATION\_TERM = ‘Code’. The value domain restriction is also created for the SC. Since this is a new SC that does not exist in the CDTC, the value domain restriction is more involving where new records in the CDT\_SC\_AWD\_PRI and CDT\_SC\_AWD\_PRI\_XPS\_MAP need to be created even though this is an SC of a BDT. More about this later.

**Example 4**

The IDType has its DEN = ‘Identifier. Type’. In fact, any ‘ID’ token in a camel case name will be expanded to ‘Identifier’ in the repository.

## xsd:complexType with complex content (ACC) or xsd:group definitions (ACC & ASCCP)

It is important to note that an xsd:complexType with complex content is defined either by the xsd:complexType/xsd:sequence structure or xsd:complexType/xsd:complexContent/xsd:extension/xsd:sequence structure. Type definitions in both cases are imported into the ACC table; however, the latter will have the BASED\_ACC\_ID column populated. The xsd:group definitions are also imported into the ACC table; and a corresponding ASCPP record is also created. The column OAGIS\_COMPONENT\_TYPE differentiates the xsd:complexType and xsd:group (the DEN of the xsd:group also has the pattern ‘Xyz Group. Details’ such as ‘Free Form Text Group. Details’).

This kind of type definitions is found in the Components.xsd, Meta.xsd, Extensions.xsd, and schemas within the BODs and Nouns folders (including the platform’s BODs and Nouns folders).

In OAGIS, such type and group definitions have children (associations), which can be represented as an xsd:element reference, an xsd:group reference, a local xsd:element, or an xsd:attribute. A child xsd:element or xsd:group reference is imported into the ASCC or BCC table as described in 5.8. A child local xsd:element is imported into the ASCC and ASCCP table as described in 5.9, while a child xsd:attribute is imported into the BCC and BCCP table as described in 5.6.

**Example 1**

The snippet of type definition from the Components.xsd below is imported as described below.

<xsd:complexType name="PartyIdentificationType" id="oagis-id-79cb3c416f1d4f3a9df5c3cac706e117">

<xsd:sequence>

<xsd:element ref="ID" id="oagis-id-4ce60ce679fe4828a505a9d7525d8e7c" minOccurs="0" maxOccurs="unbounded"/>

<xsd:element ref="PartyIDSet" id="oagis-id-0211c7159753415098692bf944bcefbf" minOccurs="0" maxOccurs="unbounded"/>

<xsd:element ref="TaxIDSet" id="oagis-id-c1bea7eb473a481bb300a1ae5dd2f134" minOccurs="0" maxOccurs="unbounded"/>

…..

</xsd:sequence>

</xsd:complexType>

A record is created in the ACC table for the type with its OBJECT\_CLASS\_TERM = ‘Party Identification’ and DEN = ‘Party Identification. Details’, OAGIS\_COMPONENT\_TYPE = 1 (meaning Semantics). In fact, DEN of an ACC is simply the OBJECT\_CLASS\_TERM suffixed with ‘. Details’.

**Example 2**

<xsd:complexType name="PartyBaseType" abstract="true" id="oagis-id-190ef5e85a8c4edc9f134e0458faad5c" mixed="false">

<xsd:complexContent>

<xsd:extension base="PartyIdentificationType">

<xsd:sequence minOccurs="1" maxOccurs="1">

<xsd:element ref="AccountID" id="oagis-id-41d0653ca3694ddcbf9b720885a9461f" minOccurs="0" maxOccurs="unbounded">

<xsd:annotation>

<xsd:documentation source="http://www.openapplications.org/oagis/10/platform/2">Identifier of the account</xsd:documentation>

</xsd:annotation>

</xsd:element>

……………

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

In this case, a record in the ACC table is created with DEN = ‘Party Base. Details’. It also has its BASED\_ACC\_ID column pointed to the ACC record in Example 1. This ACC record also has its IS\_ABSTRACT column switched to TRUE (the default is FALSE) and OAGIS\_COMPONENT\_TYPE = 0 (meaning Base, because it has ‘Base’ in the name).

Note that the mixed flag is not implemented. Mixed content is never allowed in OAGIS.

**Example 3**

<xsd:group name="ShippingWeightAndVolumeGroup" id="oagis-id-c63646ed6d41428296611baff9287fc3">

<xsd:sequence>

<xsd:element ref="DunnageWeightMeasure" id="oagis-id-38b3eee16898428692c46efa3ed50b9e" minOccurs="0" maxOccurs="1"/>

<xsd:element ref="TareWeightMeasure" id="oagis-id-a3f29aa4c6a146c19557b2f6bb5ab100" minOccurs="0" maxOccurs="1"/>

…..

</xsd:sequence>

</xsd:group>

The xsd:group definition above is imported into the ACC table with OBJECT\_CLASS\_TERM = ‘Shipping Weight and Volume Group’, DEN = ‘Shipping Weight And Volume Group. Details’. The fact that it is a group is captured by its OAGIS\_COMPONENT\_TYPE = 4 (meaning Semantic Group, see the database schema for a complete list of values and their meanings). An ASCCP record is also created with the PROPERTY\_TERM = ‘Shipping Weight And Volume Group’ and DEN = ‘Shipping Weight And Volume Group. Shipping Weight And Volume Group’.

**Example 4**

<xsd:complexType name="PartyType" abstract="false" id="oagis-id-82916fbf485b4586b85bed61a3ae0066" mixed="false">

<xsd:complexContent>

<xsd:extension base="PartyBaseType">

<xsd:sequence minOccurs="1" maxOccurs="1">

<xsd:element name="Extension" type="PartyExtensionType" id="oagis-id-3eb53be7c3264d8ea42e3215bce196fa" minOccurs="0" maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

The xsd:complexType definition above is imported into the ACC table with OBJECT\_CLASS\_TERM = ‘Party’, DEN = ‘Party. Details’, BASED\_ACC\_ID pointing to the ACC imported in Example 2, and OAGIS\_COMPONENT\_TYPE = 1 (meaning Semantics).

Import of contents within the type and group definitions within these three examples will be illustrated in subsequent sections.

## Global xsd:element (BCCP) or xsd:attribute declaration using a BDT (BCCP & BCC)

A global xsd:element declaration using a BDT is a global xsd:element that uses the type definitions imported in section 5.1 and 5.4. Similarly, an xsd:attribute declaration using a BDT is an xsd:attribute defined in an ACC that uses the type definitions imported in section 5.1 and 5.4.

Global xsd:element declarations using a BDT are mostly found in Fields.xsd, while some exist in Meta.xsd.

The xsd:attribute declarations using a BDT are mostly found in Nouns, BODs, and Components.xsd schemas, while a few exist in Meta.xsd.

The occurrences of these declarations are imported into the BCCP table. Additionally, entries in the BCC table are created for each xsd:attribute. In other words, this kind of xsd:attribute declarations is treated as both a declaration an a usage (association). However, the xsd:attribute declarations with the same name and BDT has only one entry in the BCCP table. More specifically, the id attribute of the xsd:attribute is used as an association GUID in the BCC table, while the GUID of the corresponding BCCP is newly generated by the import routine.

**Note**: xsd:attribute that are imported as a BCCP are those whose type is a BDT. An xsd:attribute whose type is from the XBT (this includes the XML schema built-in type and those imported by section 5.2), CODE\_LIST, or AGENCY\_ID\_LIST table is imported as an SC.

**Example 1**

<xsd:element name="LanguageCode" type="LanguageCodeType" nillable="true" id="oagis-id-4419527f02ca4a90bd139c20e084b562"/>

A BCCP record is created for the global xsd:element declaration above. It has PROPERTY\_TERM = ‘Language Code’, REPRESENTATION\_TERM = ‘Code’, DEN = ‘Language Code. Code’, IS\_NILLABLE = TRUE, and BDT\_ID pointing to the LanguageCodeType record in the DT table. The representation term is taken from data type term of the associated data type. As can be seen in the DEN, it is important point to note that the truncation rule in the CCTS is not adopted when combining the property term and representation term to make a DEN for the BCCP. If the element has a default value, the DEFAULT\_VALUE column would be filled.

**Example 2**

<xsd:complexType name="BusinessObjectDocumentType" id="oagis-id-2783857358f145e799471461f5192fa7">

<xsd:sequence>

<xsd:element ref="ApplicationArea" id="oagis-id-c2cb6823837d4149b32aefb8fd4120cd"/>

</xsd:sequence>

<xsd:attribute name="releaseID" type="NormalizedStringType" use="required" id="oagis-id-0e403050beea4692a5b92eacf5c81b41"/>

…..

<xsd:attribute name="languageCode" type="LanguageCodeContentType" use="optional" default="en-US" id="oagis-id-c0e5355ae62649a4b5e00fdc79144568"/>

</xsd:complexType>

In the ACC type definition above, there are two xsd:attribute declarations, namely releaseID and languageCode. Let’s assume that there has been no BCCP created for an xsd:element or xsd:attribute with the property term ‘Release Identifier’ with the type NormalizedStringType. Therefore, a BCCP record is created with PROPERTY\_TERM = ‘Release Identifier’, REPRESENTATION\_TERM = ‘Text’, DEN = ‘Release Identifier. Text’, BDT\_ID pointing to the NormalizedStringType record in the DT table. The REPRESENTATION\_TERM is ‘Text’ because the NormalizedStringType is a lineage of the ‘Text. Type’ CDT whose data type term is ‘Text’.

In Example 1, a BCCP record with the same property term, ‘Language Code’, was already created. However, another BCCP record with the same property term has to be created because they have different BDTs. In this case, the BCCP record has the same PROPERTY\_TERM = ‘Language Code’, REPRESENTATION\_TERM = ‘Code’, and DEN = ‘Language Code. Code’, but the BDT\_ID pointing to the LanguageCodeContentType record in the DT table. The REPRESENTATION\_TERM is ‘Code’ because the LanguageCodeContentType is a lineage of the ‘Code. Type’ CDT whose data type term is ‘Code’.

BCC records are also created to represent the associations to these two xsd:attribute BCCPs. The first BCC record has FROM\_ACC\_ID pointing to the ‘Business Object Document. Details’ ACC record, TO\_BCCP\_ID pointing to the first ‘Release Identifier. Text’ BCCP record created above, DEN = ‘Business Object Document. Release Identifier. Text’. The other BCC record has FROM\_ACC\_ID pointing to the ‘Business Object Document. Details’ ACC record, TO\_BCCP\_ID pointing to the second Language Code. Code’ BCCP record created above, DEN = ‘Business Object Document. Language Code. Code’. Additionally, the BCC record has its DEFAULT\_VALUE = ‘en-US’ (notice that for this case of BCCP attribute, the default value is imported into the BCC record as opposed to the BCCP record as in the case of BCCP element).

## Global xsd:element declaration using the ACC (ASCCP)

This is a global xsd:element declaration whose type points to the type imported per section 5.5. Such xsd:element declaration is imported into the ASCCP table (with the REUSABLE\_INDICATOR = TRUE as opposed to the local xsd:element in 5.9 which has REUSABLE\_INDICATOR = FALSE).

**Example 1**

<xsd:element name="Party" type="PartyType" abstract="false" nillable="false" id="oagis-id-fe283b0b4cc641c5b709f4375d4ad090"/>

<xsd:element name="PartyIDSet" type="IDSetType" id="oagis-id-baf3407368e849338ea10b5ec6902766"/>

<xsd:element name="TaxIDSet" type="IDSetType" id="oagis-id-9ea03290b7c7478bb7409699f73e2c41"/>

Three records in the ASCCP table are created corresponding to the global element declarations above. The first record has PROPERTY\_TERM = ‘Party’, ROLE\_OF\_ACC\_ID pointing to the ‘Party. Details’ ACC imported in Example 4 of section 5.5, DEN = ‘Party. Party’ (because DEN is a concatenation of its PROPERTY\_TERM and the ‘Party. Details’ ACC’s OBJECT\_CLASS\_TERM).

The second record has PROPERTY\_TERM = ‘Party Identifier Set’, ROLE\_OF\_ACC\_ID pointing to the ‘Identifier Set. Details’ ACC, DEN = ‘Party Identifier Set. Identifier Set’. Notice that the ‘ID’ is unabbreviated to ‘Identifier’; and there is no truncation in the property and representation term. The third record uses the same ACC. It is imported in a similar fashion with the PROPERTY\_TERM = ‘Tax Identifier Set’ and DEN = ‘Tax Identifier Set. Identifier Set’.

## xsd:element or xsd:group reference (ASCC & BCC)

The xsd:element reference is an association; hence, it is imported into either the BCC table when referring to a BCCP or the ASCC table when referring to an ASCCP. The xsd:group reference is also an association, but a group is always an aggregated component so it is always imported into the ASCC table.

Because some entries in the BCC table is a result of xsd:attribute as in section 5.6 while some is a result of xsd:element reference as in this section, the BCC table has an ENTITY\_TYPE column that indicates whether a particular BCC is an xsd:element or an xsd:attribute. (It may be worth noting that this indicator is not kept in the BCCP table so that the BCCP remains independent of expression, which is the spirit of CCS. However, certain BCCPs cannot be used as xsd:attribute because it has xsd:attribute in itself, the SC.)

**Example 1**

In this example, we use Example 1 from section 5.5 with a slight addition. This example does not correspond to the real OAGIS content.

<xsd:complexType name="PartyIdentificationType" id="oagis-id-79cb3c416f1d4f3a9df5c3cac706e117">

<xsd:sequence>

<xsd:element ref="ID" id="oagis-id-4ce60ce679fe4828a505a9d7525d8e7c" minOccurs="0" maxOccurs="unbounded"/>

<xsd:element ref="PartyIDSet" id="oagis-id-0211c7159753415098692bf944bcefbf" minOccurs="0" maxOccurs="unbounded"/>

<xsd:element ref="TaxIDSet" id="oagis-id-c1bea7eb473a481bb300a1ae5dd2f134" minOccurs="0" maxOccurs="unbounded"/>

<xsd:group ref="ShippingWeightAndVolumeGroup" id="oagis-id-47d8f7eb473a481bb300a1ae5dd2f134" minOccurs="1" maxOccurs="1"/>

…..

</xsd:sequence>

</xsd:complexType>

In the example ‘Party Identification. Details’ ACC above, there is one BCC for the ID element reference and three ASCCs – two for the PartyIDSet and TaxIDSet element references and the other for the ShippingWeightAndVolumeGroup group reference.

A record in the BCC table is created with DEN = ‘Party Identification. Identifier. Identifier’ (basically the concatenation of the object class term of the ACC and the DEN of the BCCP, being ‘Identifier. Identifier’).

An ASCC record for the PartyIDSet, TaxIDSet, and ShippingWeightAndVolumeGroup is created with DEN = ‘Party Identification. Party Identifier Set. Identifier Set’, ‘Party Identification. Tax Identifier Set. Identifier Set’, ‘Party Identification. Shipping Weight And Volume Group. Shipping Weight And Volume Group’.

## Local xsd:element with global type (ASCCP & ASCC)

In OAGIS, there are two kinds of local elements one is the DataArea and the other is Extension. Each BOD schema has a local DataArea element, while a number of ACCs has an Extension local element (whose type is defined in Extensions.xsd). It should be noted that OAGIS also defined named Extension global element corresponding to each named Extension type (ACC) though these are never used. These are imported as ASCCPs in the same way as described in section 5.7.

Each occurrence of a local xsd:element results in one entry in the ASCCP table and another entry in the ASCC table. The id attribute of the xsd:element is used as the GUID of the ASCC, while the GUID of the ASCCP is generated by the import routine.

**Example 1**

<xsd:complexType name="SyncBOMType" id="oagis-id-d048f400279b4d8e8d6bd6047e1c80b2">

<xsd:complexContent>

<xsd:extension base="BusinessObjectDocumentType">

<xsd:sequence>

<xsd:element name="DataArea" type="SyncBOMDataAreaType" id="oagis-id-415bc79809844caea49a49366a0f4827"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

In the SyncBOMType definition (‘Sync BOM. Details’ ACC) snippet above, the DataArea is declared as a local element that uses the complex type SyncBOMDataAreaType (i.e., the ‘Sync BOM Data Area. Details’ ACC). For the DataArea local element, an ASCCP record is created with PROPERTY\_TERM = ‘Data Area’, ROLE\_OF\_ACC\_ID pointing to the record in the ACC table corresponding to the ‘Sync BOM Data Area. Details’ ACC, DEN = ‘Data Area. Sync BOM Data Area’, and the GUID column is filled with a newly generated unique value. Then, an ASCC record is created for its association with the ‘Sync BOM. Details’ ACC. The ASCC record has DEN = ‘Sync BOM. Data Area. Sync BOM Data Area’ and GUID = ‘oagis-id-415bc79809844caea49a49366a0f4827’. Due to this nature, there are many ASCCPs with the same ‘Data Area’ property term but different GUIDs.

Example 2

<xsd:complexType name="HeaderType" abstract="true" id="oagis-id-e65a94696836447da647a0f899af4988" mixed="false">

<xsd:complexContent>

<xsd:extension base="HeaderBaseType">

<xsd:sequence minOccurs="1" maxOccurs="1">

<xsd:element name="Extension" type="HeaderExtensionType" id="oagis-id-c60cec63c426441eaf64f222169e6d33" minOccurs="0" maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

In the HeaderType definition (‘Header. Details’ ACC) snippet above, the Extension element is declared as a local element that uses the complex type HeaderExtensionType (‘Header Extension. Details’ ACC). For the Extension local element, an ASCCP record is created with PROPERTY\_TERM = ‘Extension’, ROLE\_OF\_ACC\_ID pointing to the ‘Header Extension. Details’ ACC, DEN = ‘Extension. Header Extension’, and the GUID column is filled with a newly generated unique value. Then, an ASCC record is created for its association with the ‘Header. Details’ ACC. The ASCC record has DEN = ‘Header. Extension. Header Extension’ and GUID = ‘oagis-id-c60cec63c426441eaf64f222169e6d33’. Due to this nature, there are many ASCCPs with the same ‘Extension’ property term but different GUIDs.

## xsd:any

Two entities are created to represent the xsd:any, the ‘Any Structured Content. Details’ ACC and the ‘Any Property. Any Structured Content’ ASCCP. The ACC represents the xsd:any with any namespace and strict content processing. The is the only xsd:any variation used in OAGIS 10.2 (in the future a qualified ACC based on the ‘Any Structured Content. Details’ ACC may be created to represent another variation, e.g., ‘Boeing\_ Any Structured Content. Details’ to represent a xsd:any that restricts its content to a particular Boeing Co.’s namespace). For a usage of the xsd:any in the OAGIS schema, an ASCC record to the ASCCP will be created.

**Example 1**

<xsd:complexType name="SignatureType" id="oagis-id-39f8794875974287b462ef4e501d1561">

<xsd:annotation>

<xsd:documentation source="http://www.openapplications.org/oagis/10/platform/2">Allows any digital Signatures to be added to the Instance of the BOD Instance in order to sign for any portion of the BOD content.</xsd:documentation>

</xsd:annotation>

<xsd:sequence>

<xsd:any namespace="##any" id="oagis-id-44622f35a5b547da899fea2d7753bf96" minOccurs="0"/>

</xsd:sequence>

<xsd:attribute name="qualifyingAgencyId" type="NormalizedStringType" use="optional" id="oagis-id-329bb34257c141368a9dd472862fd4c0"/>

</xsd:complexType>

In the ‘Signature. Details’ ACC above, it contains a usage of the xsd:any with any namespace and (by default) strict content processing. Therefore, an ASCC record is created with DEN = ‘Signature. Any Property. Any Structured Content’, TO\_ASCCP\_ID referring to the ‘Any Property. Any Structured Content’ ASCCP, and GUID = ‘oagis-id-44622f35a5b547da899fea2d7753bf96’.

## Sequencing key assignment

The ASCC and BCC tables have a SEQ\_KEY column, which represents the order of associations under the xsd:sequence. The sequencing key of a child in the type or group’s xsd:sequence always start at 1 for the first child and increasing to 2, 3, and so on for subsequent children. That is when a type is derive from (i.e., extends) another type, the sequencing key of the children in the derived type restarts at 1 again.

## Documentation

Import of the content within the xsd:documentation has not yet been formalized in this version of repository. Further work is needed to consistently import the documentation. In most cases, all the content under the xsd:documentation is tag is imported into the DEFINITION column of the table where the component is imported. In addition, only one xsd:documentation under the xsd:annotation is considered.

**Example 1**

Taking the documentation of the ‘Signature. Details’ ACC in the Example 1 of section 5.10 above, the ACC record has its DEFINTION column populated as ‘Allows any digital Signatures to be added to the Instance of the BOD Instance in order to sign for any portion of the BOD content.’

**Example 2**

<xsd:complexType name="LicenseBaseType" abstract="false" id="oagis-id-a93c02eb079940889cd5acf220b1217f" mixed="false">

<xsd:annotation>

<xsd:documentation source="http://www.hr-xml.org" xml:lang="en">

<ccts\_CategoryCode>ABIE</ccts\_CategoryCode>

<ccts\_DictionaryEntryName>License. Details</ccts\_DictionaryEntryName>

<ccts\_DefinitionText>Authoritative permission to hold a certain status or to do certain things, e.g. to practice some trade or profession. (OED license, n. a. A formal, usually a printed or written permission from a constituted authority to do something. b. The document embodying such a permission.)</ccts\_DefinitionText>

</xsd:documentation>

</xsd:annotation>

<xsd:complexContent>

<xsd:extension base="IdentificationType">

……..

……..

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

In the example above, the DEFINITION column of the ‘License Base. Details’ ACC is populated with the whole XML content that is highlighted under the xsd:documentation element.

## Value domain restriction

This section describes how a value domain of a specific expression is assigned to a BDT or a BDT SC. A value domain can be a built-in type (see definition in section 3.3) or a values list. An example of an expression is XML schema. The built-in types for the XML schema expression are stored in the XBT table. The table below shows some example data in the XBT table (not all columns shown).

**XBT**

|  |  |  |
| --- | --- | --- |
| XBT\_ID | NAME | BUILTIN\_TYPE |
| 1 | String | xsd:string |
| 2 | normalized string | xsd:normalizedString |
| 3 | token | xsd:token |
| 4 | xbt boolean true false | XbtBooleanTrueFalseType |

### Value domain restriction for a BDT

The main table responsible for the value domain assignment to the BDT is the BDT\_PRI\_RESTRI table. The value domain may come from the CDT\_AWD\_PRI\_XPS\_TYPE\_MAP (for built-in type assignment), CODE\_LIST, or AGENCY\_ID\_LIST table.

A BDT is always a derivative of a CDT; therefore, it inherits CDT allowed primitive per the CDTC specification. For example, TextType\_0VCBZ5 shown in Example 1 of section 5.1 is a derivative of the ‘Text. Type’ CDT.

According to the CDTC, the CDT primitives allowed for the ‘Text. Type’ CDT (content component) are NormalizedString, String, and Token. This information is kept in the CDT\_AWD\_PRI table as illustrated below.

**CDT\_AWD\_PRI**

|  |  |  |  |
| --- | --- | --- | --- |
| CDT\_AWD\_PRI\_ID | CDT\_ID | CDT\_PRI\_ID | IS\_DEFAULT |
| 1 | Text. Type | String | TRUE |
| 2 | Text. Type | NormalizedString | FALSE |
| 3 | Text. Type | Token | FALSE |
| 4 | Code. Type | String | FALSE |
| 5 | Code. Type | NormalizedString | FALSE |
| 6 | Code. Type | Token | TRUE |

The mapping between the CDT primitives and built-in types is stored in the CDT\_AWD\_PRI\_XPS\_TYPE\_MAP (XPS for expression). This table is indeed a three-way mapping across the CDT, CDT primitive, and the built-in type (instead of simply between the CDT primitive and the built-in type[[1]](#footnote-1)). The reason for this is to handle the case when more than one CDT have the same CDT primitive but different maps to built-in types. The table below shows some example data (note that in example data throughout this section human understandable data is shown instead of keys in some column for better understandability).

**CDT\_AWD\_PRI\_XPS\_TYPE\_MAP**

|  |  |  |
| --- | --- | --- |
| CDT\_AWD\_PRI\_XPS\_TYPE\_MAP\_ID | CDT\_AWD\_PRI\_ID | XBT\_ID |
| 1 | 1 | xsd:string |
| 2 | 2 | xsd:normalizedString |
| 3 | 3 | xsd:token |
| 4 | 4 | xsd:string |
| 5 | 5 | xsd:normalizedString |
| 6 | 6 | xsd:token |

Because a BDT is a derivative of a CDT, in principle its value domain must be restricted to a subset of those XBTs allowed on the CDT. In certain cases, the subset is a defined list of values which are specified by a code list or an agency ID list. Consequently, the BDT\_PRI\_RESTRI table has three columns that allow these three ways of specifying the value domain including the CDT\_AWD\_PRI\_XPS\_TYPE\_MAP\_ID, CODE\_LIST\_ID, AGENCY\_ID\_LIST\_ID. The first points to a record in the CDT\_AWD\_PRI\_XPS\_TYPE\_MAP table shown above, the second points to a record in the CODE\_LIST table, and the last points to a record in the AGENCY\_ID\_LIST table. The example below shows the value domain restriction for the TextType\_0VCBZ5 BDT. It points to the second record in the CDT\_AWD\_PRI\_XPS\_TYPE\_MAP example above, because it is a based on the xsd:normalizedString.

**BDT\_PRI\_RESTRI**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| BDT\_PRI\_RESTRI\_ID | BDT\_ID | CDT\_AWD\_PRI\_XPS\_TYPE\_MAP\_ID | CODE\_LIST\_ID | AGENCYID\_LIST\_ID | IS\_DEFAULT |
| 1 | Text\_0VCBZ5. Type | 2 |  |  | TRUE |
| 2 | Operator\_ Code. Type | 6 |  |  | TRUE |
| 3 | Operator\_ Code. Type |  | oacl\_OperatorCode |  | FALSE |

The examples in BDT\_PRI\_RESTRI table above also includes a value domain restriction for the ‘Operator\_ Code. Type’ BDT as well. The ‘Operator\_ Code. Type’ is defined as shown below. Because it is based on the ‘Operator\_ Code Content. Type’ BDT, which in turn is based on the ‘oacl\_OperatorCode’ code list, it has two entries in the BDT\_PRI\_RESTRI table. The first one points to the xsd:token that is the default; and the other points to the ‘oacl\_OperatorCode’ code list. The xsd:token is one of the value domains because of the extensibility nature of OAGIS code lists (when a BIE is derived, the value domain can be restricted to only the code list).

<xsd:complexType name="OperatorCodeType" id="oagis-id-16afc52317bb4adca1dbaeb72d4c7190">

<xsd:simpleContent>

<xsd:extension base="OperatorCodeContentType">

<xsd:attribute name="listID" type="xsd:normalizedString" use="optional" id="oagis-id-3d8eec0a38b94bc69d9d40d9d224d689"/>

<xsd:attribute name="listAgencyID" type="clm63055D08B\_AgencyIdentificationContentType" use="optional" id="oagis-id-ac4eed3fe8c24f469cfc1f47ca8f63ac">

<xsd:attribute name="listVersionID" type="xsd:normalizedString" use="optional" id="oagis-id-2d34df84eb69473592137596e3f6de36"/>

</xsd:extension>

</xsd:simpleContent>

</xsd:complexType>

### Value domain restriction for a standard SC from CDTC

The mechanism for the value domain restriction for the SC that is derived from the standard, is very much similar to the case of the BDT. It is only that the set of tables involved differs. In this case, the main table is BDT\_SC\_PRI\_RESTRI. When the value domain is a built-in type, a column in the BDT\_SC\_PRI\_RESTRI points to a record in the CDT\_SC\_AWD\_PRI\_XPS\_TYPE\_MAP table. The value domain can be a code list or an agency ID list as in the case of the BDT. In such case, the BDT\_SC\_PRI\_RESTRI has columns pointing, respectively, to a record in the CODE\_LIST or the AGENCY\_ID\_LIST table.

### Value domain restriction for an OAGIS defined SC

Some SCs are newly defined in OAGIS. The difference in this case from the case of standard SC above is that a new record or records are needed for the new SC in the CDT\_SC\_AWD\_PRI\_XPS\_TYPE\_MAP table in order to be able to assign the built-in type (in the BDT\_SC\_PRI\_RESTRI table). However, to have a new record in that table new records are also needed in the CDT\_SC\_AWD\_PRI for the DT\_SC record corresponding to the new SC.

The key to creating records in the CDT\_SC\_AWD\_PRI for the new SC is to determine CDT primitives allowed for that new SC. The import routine determines the allowed CDT primitives based on the representation term of the SC. The representation term of the new SC is determined using the last token of the name of the SC, but if the last token is not one of the valid CDTC representation terms then the representation term shall be ‘Text’. E.g., if the name is ‘sequenceNumber’, the representation term is ‘Number’, if the name is ‘expressionLanguage’, the representation term is ‘Text’. The allowed CDT primitives are then derived from the allowed primitives of a CDT whose data type term matches the representation term. The correspondence between the data type term (or representation term) and the allowed CDT primitives per the CDTC specification is summarized in Appendix 2 for reference.

# Mapping Summary

|  |  |  |
| --- | --- | --- |
| XML Schema Construct | CCS Entity | Affected Tables |
| Global xsd:simpleType with no xsd:enumeration and no xsd:pattern | Business Data Type | DT, DT\_SC, BDT\_PRI\_RESTR, BDT\_SC\_PRI\_RESTR |
| Global xsd:complexType with xsd:simpleContent and its xsd:attribute |
| Global xsd:simpleType with no xsd:enumeration but with xsd:pattern | NA. This is expression specific. | XBT |
| Global xsd:simpleType with xsd:enumeration | NA. This is a list of values. | CODE\_LIST, AGENCY\_ID\_LIST, CODE\_LIST\_VALUE, AGENCY\_ID\_LIST\_VALUE |
| Global xsd:complexType[count(xsd:sequence) = 1] | ACC | ACC |
| Global xsd:complexType[count(xsd:complexContent/ xsd:extension/xsd:sequence) = 1] |
| Global xsd:group declaration | ACC and ASCCP | ACC, ASCCP |
| Global xsd:element declaration | BCCP or ASCCP | BCCP, ASCCP |
| Local xsd:element (with global type) declaration | ASCCP and ASCC | ASCCP, ASCC |
| xsd:attribute in an ACC | BCC and BCCP | BCC, BCCP |
| xsd:element reference | BCC or ASCC | BCC, ASCC |
| xsd:group reference | ASCC | ASCC |
| xsd:any | ‘Any Structured Content’ ACC,  ‘Any Property’ ASCCP, ASCC | ACC, ASCCP, ASCC |

Appendix 1: CDT Primitive and Built-in Type Map

|  |  |
| --- | --- |
| CDT Primitive | XSD Built-in type |
| Binary | xsd:base64Binary |
| Binary | xsd:hexBinary |
| Boolean | xbt\_BooleanTrueFasleType |
| Decimal | xsd:decimal |
| Double | xsd:double |
| Double | xsd:float |
| Float | xsd:float |
| Integer | xsd:integer |
| Integer | xsd:nonNegativeInteger |
| Integer | xsd:positiveInteger |
| NormalizedString | xsd:normalizedString |
| String | xsd:string |
| TimeDuration | xsd:token |
| TimeDuration | xsd:duration |
| TimePoint | xsd:token |
| TimePoint | xsd:dateTime |
| TimePoint | xsd:date |
| TimePoint | xsd:time |
| TimePoint | xsd:gYearMonth |
| TimePoint | xsd:gYear |
| TimePoint | xsd:gMonthDay |
| TimePoint | xsd:gDay |
| TimePoint | xsd:gMonth |
| Token | xsd:token |

Appendix 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DT Term or Rep Term | CDT Primitives |  | DT Term or Rep Term | CDT Primitives |  | DT Term or Rep Term | CDT Primitives |
| Amount | Integer |  | Name | Token |  | Rate | Decimal |
| Amount | Float |  | Name | String |  | Ratio | String |
| Amount | Double |  | Name | NormalizedString |  | Ratio | Integer |
| Amount | Decimal |  | Number | Integer |  | Ratio | Float |
| Binary Object | Binary |  | Number | Float |  | Ratio | Double |
| Code | Token |  | Number | Double |  | Ratio | Decimal |
| Code | String |  | Number | Decimal |  | Sound | Binary |
| Code | NormalizedString |  | Ordinal | Integer |  | Text | Token |
| Date | TimePoint |  | Percent | Integer |  | Text | String |
| Date Time | TimePoint |  | Percent | Float |  | Text | NormalizedString |
| Duration | TimeDuration |  | Percent | Double |  | Time | TimePoint |
| Graphic | Binary |  | Percent | Decimal |  | Value | Token |
| Identifier | Token |  | Picture | Binary |  | Value | String |
| Identifier | String |  | Quantity | Integer |  | Value | NormalizedString |
| Identifier | NormalizedString |  | Quantity | Float |  | Value | Integer |
| Indicator | Boolean |  | Quantity | Double |  | Value | Float |
| Measure | Integer |  | Quantity | Decimal |  | Value | Double |
| Measure | Float |  | Rate | Integer |  | Value | Decimal |
| Measure | Double |  | Rate | Float |  | Video | Binary |
| Measure | Decimal |  | Rate | Double |  |  |  |

1. The mapping between the CDT primitives and built-in types is provided in Appendix 1 for reference. [↑](#footnote-ref-1)