Semantic Refinement Tool:

Software Design Document

Editor: Yunsu Lee, NIST/UMBC

Creation Date: 05/12/2014

Last Updated: 12/9/2014 11:46 AM

Draft / Version: 0.1

Status: IN PROCESS

Contributors:

Serm Kulvatunyou, NIST

Jaehun Lee, NIST/POSTECH

Michael Rowell, Oracle

Steffen Fohn, ADP

David Connelly, OAG

Ralph Hertlien, OAG/Boeing

Pat O’Connor, Infor

Ian Hedges, E2OPEN

Alonso Moncayo, E2OPEN

Change Tracking

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Detail |
| 05/12/2014 | Yunsu Lee | 0.1 | Created the first draft. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Change Tracking 2](#_Toc387651765)

[Table of Contents 3](#_Toc387651766)

[1. Introduction 5](#_Toc387651767)

[1.1 Purpose 5](#_Toc387651768)

[1.2 Scope 5](#_Toc387651769)

[1.3 Definitions, Acronyms and Abbreviations 5](#_Toc387651770)

[1.4 References 5](#_Toc387651771)

[2. Architectural Goals and Constraints 6](#_Toc387651772)

[2.1 Technical Platform 6](#_Toc387651773)

[2.2 Security 6](#_Toc387651774)

[2.3 Persistence 6](#_Toc387651775)

[2.4 Reliability/Availability 6](#_Toc387651776)

[2.5 Performance 6](#_Toc387651777)

[2.6 Internationalization 6](#_Toc387651778)

[3. Use-Case View 7](#_Toc387651779)

[3.1 Import OAGIS 10 Model into the Database 7](#_Toc387651780)

[3.1.1 Import OAGIS 10 Model into the Database 7](#_Toc387651781)

[3.1.2 Verify the OAGIS 10 Model import 7](#_Toc387651782)

[3.2 BIEs Management 7](#_Toc387651783)

[3.2.1 Create a top-level ABIE 7](#_Toc387651784)

[3.2.1.1 Expand/Collapse a descendant ASBIE 7](#_Toc387651785)

[3.2.1.2 Expand/Collapse a descendant BBIE 7](#_Toc387651786)

[3.2.1.3 Customize a child ASBIE 7](#_Toc387651787)

[3.2.1.4 Customize a child BBIE 7](#_Toc387651788)

[3.2.2 Save a top-level ABIE 7](#_Toc387651789)

[3.2.3 View and Edit a top-level ABIE 7](#_Toc387651790)

[3.2.4 Create a top-level ABIE by copy 7](#_Toc387651791)

[3.3 Generate an OAGIS Expression 7](#_Toc387651792)

[3.3.1 Generate a standalone XML Schema for a top-level ABIE 7](#_Toc387651793)

[3.4 Code List Management 7](#_Toc387651794)

[3.4.1 Create a new blank BIE code list 7](#_Toc387651795)

[3.4.2 Create a new BIE code list by restriction 7](#_Toc387651796)

[3.4.3 Save a working BIE code list 7](#_Toc387651797)

[3.4.4 Edit a BIE code list 7](#_Toc387651798)

[3.5 Manage CCs and DTs 7](#_Toc387651799)

[3.5.1 View CCs 7](#_Toc387651800)

[4. Logical View 8](#_Toc387651801)

[4.1 Overview 8](#_Toc387651802)

[4.2 Design Packages 8](#_Toc387651803)

[5. Process View 9](#_Toc387651804)

[6. Deployment View 10](#_Toc387651805)

[7. Implementation View 11](#_Toc387651806)

[7.1 Overview 11](#_Toc387651807)

[7.2 Layers 11](#_Toc387651808)

[7.2.1 Presentation Layer 11](#_Toc387651809)

[7.2.2 Control layer 11](#_Toc387651810)

[7.2.3 Resource Layer 11](#_Toc387651811)

[7.2.4 Domain Layer 11](#_Toc387651812)

[7.2.5 Common Layer 11](#_Toc387651813)

[8. Data View 12](#_Toc387651814)

# Introduction

## Purpose

## Scope

## Definitions, Acronyms and Abbreviations

## References

# Architectural Goals and Constraints

## Technical Platform

AAA

## Security

## Persistence

## Reliability/Availability

## Performance

## Internationalization

# Use-Case View

## Import OAGIS 10 Model into the Database

Create a default user with user id “OAGISUser” with password the same as the user id.

For all XPATH expressions below, if the node does not exist, the value should be blank (not a “null” string as Java may convert a non-existing node to a “null” string) unless otherwise specified.

### Import OAGIS 10 Model into the Database

#### Create an OAGIS user

Create a default OAGIS user. All OAGIS model content will belong to this user. Populate the User table as follows.

User\_ID = Auto-generate database key.

User\_Name = “oagis”

Password = “oagis”

Name = “Open Applications Group Developer”

Organization = “Open Applications Group”

#### Populate CDT data

Create a script to populate Core Data Types (CDTs) data based on the CCTS Data Type Catalog V3 (CCTS DTC3). Data in this section will not be coming from any OAGIS schema.

##### Populate the XSD\_BuiltIn\_Type table

Populate this table with W3C XSD built-in datatypes from the types hierarchy in the figure below (from <http://www.w3.org/TR/xmlschema-2/#built-in-datatypes>). At this point only bring in the types in the polygon.

XSD\_BuiltIn\_Type\_ID = Auto-generate database key.

BuiltIn\_Type = Take the names from the types hierarchy and prefix with ‘xsd:’, e.g., ‘xsd:token’.

Name = Take the names from the types hierarchy and apply a few separation patterns as follows.

* Use all lower case letters.
* Generally separate the camel case with a space instead, e.g., ‘positiveInteger’ -> ‘positive integer’.
* g = Gregorian

Subtype\_Of\_XSD\_BuiltIn\_Type\_ID = Self-referenced foreign key to the XSD\_BuiltIn\_Type\_ID of the parent type in the hierarchy.



##### ***Populate the CDT\_Primitive table***

Populate the CDT\_Primitive table with information from the table in section 3.2.1 of CCTS DTC3. Use the Name column for the CDT\_Primitive.Name.

##### ***Populate CDTs in the DT table***

Populate the DT table with CDT information from the CCTS DTC3 as follows.

DT\_ID = Auto-generate database key.

DT\_GUID is generated one time and then fixed.

DT\_Type = “0” (note: 0 indicates CDT).

Version\_Number = “1.0”

Previous\_Version\_DT\_ID = Leave blank.

Revision\_Type = “0” (note: 0 means NEW).

Data\_Type\_Term = Each CDT in CCTS DTC3 section 4 indicates this, e.g., “Amount”.

Qualifier = Blank.

Based\_DT\_ID = Blank.

DEN = Take the value from each CDT subsection in CCTS DTC3 section 4, e.g., “Amount. Type”.

Content\_Component\_DEN = Take the value from each CDT subsection in CCTS DTC3 section 4 indicates this, e.g., “Amount. Content”.

Definition = “CDT V3.1.” + Combine the texts from the Definition and Remarks sections from each CDT in CCTS DTC3.

Content\_Component\_Definition = Take the value from the CCTS DTC3 in the Definition column of the table in section 4.X.7.

Revision\_Documentation = Blank.

Revision\_State = “1” (note: 1 means published).

Created\_By\_User\_ID = “oagis”.

Last\_Updated\_By\_User\_ID = “oagis”.

Creation\_Timestamp = Current time.

Last\_Update\_Timestamp = Same as Creation\_Timestamp.

##### ***Populate the CDT\_Allowed\_Primitive table***

Populate the table with CDT information from the CCTS DTC3 as follows. This table capture allowed primitives of the CDT’s Content Component. Each CDT’s Content Component typically allows a few primitives, so there will be multiple records per CDT.

CDT\_Allowed\_Primitive\_ID = Auto-generate database key.

CDT\_ID = Foreign key from the DT table corresponding to the CDT being recorded.

CDT\_Primitive\_ID = Foreign key from the CDT\_Primitive table corresponding to the Allowed Primitive column in each of the CDT Content Component section/table in CCTS DTC3.

##### ***Populate the CDT\_Allowed\_Primitive\_Expression\_Type\_Map***

For each row in the CDT\_Allowed\_Primitive table, there will be zero or more rows in this table.

CDT\_Primitive\_Expression\_Type\_Map\_ID = Auto-generate database key.

CDT\_Allowed\_Primitive\_ID = Foreign key to CDT\_Allowed\_Primitive.CDT\_Allowed\_Primitive\_ID.

XSD\_BuiltIn\_Type\_ID = Foreign key to XSD\_BuiltIn\_Type. XSD\_BuiltIn\_Type\_ID

Use the following map between the CDT primitives and XSD built-in types:

Note: Double CDT Primitive is mapped to both xsd:double and xsd:float. This is under the assumption that xsd:float is technical a kind of Double, although xsd:float is not a subtype of the xsd:double in the XML schema built-in type hierarchy.

|  |  |
| --- | --- |
| **CDT Primitive** | **XSD Built-in type** |
| Binary | xsd:base64Binary |
| Binary | xsd:hexBinary |
| Boolean | xsd:boolean |
| Decimal | xsd:decimal |
| Double | xsd:double |
| Double | xsd:float |
| Float | xsd:float |
| Integer | xsd:integer |
| 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 |

All CDTs maps to all XSD built-in types according to their primitives and the primitive-to-XSD-built-in-type map in the table above except for the Date and Time CDTs. Date should be mapped to everything in the Timepoint primitive except the xsd:dateTime. Time CDT should be mapped to only xsd:token and xsd:time in the Timepoint primitive. The resulting data should represent information like the example table below.

|  |  |  |
| --- | --- | --- |
| From CDT\_Allowed\_Primitive | | XSD\_BuiltInType |
| CDT\_DEN | *CDT\_Primitive* |  |
| Amount | Decimal | xsd:decimal |
| Amount | Double | xsd:double |
| Amount | Float | xsd:float |
| Amount | Integer | xsd:integer |
| Date | TimePoint | xsd:token |
| Date | TimePoint | xsd:date |
| Date | TimePoint | xsd:gYear |
| Date | TimePoint | xsd:gYearMonth |
| Date | TimePoint | xsd:gMonthDay |
| Date | TimePoint | xsd:gDay |
| Date | TimePoint | xsd:gMonth |

##### ***Populate CDTs’ supplementary component in the DT\_SC table***

Populate the table with CDT SC information from the CCTS DTC3 as follows. Sections 4.X.8 contains information for this table.

DT\_SC\_ID = Auto-generate database key.

DT\_SC\_GUID = Generate a GUID.

Property\_Term = Take the value from the Supplementary Components subsection of each CDT section in the CCTS DTC3, e.g., “Currency”.

Representation\_Term = Take the value from the Supplementary Components subsection of each CDT section in the CCTS DTC3, e.g., “Code”.

Definition = Take the value from the Supplementary Components subsection of each CDT section in the CCTS DTC3.

Owner\_DT\_ID = Foreign key to the DT table DT\_ID column for the corresponding CDT.

Min\_Cardinality = Take the value from the Supplementary Components subsection of each CDT section in the CCTS DTC3.

Max\_Cardinality = Take the value from the Supplementary Components subsection of each CDT section in the CCTS DTC3.

Based\_DT\_SC\_ID = Blank.

##### Populate the CDT\_SC\_Allowed\_Primitive table

Populate the table according the column Allowed Primitives of the table in sections 4.X.8 of CCTS DTC3.

CDT\_SC\_Allowed\_Primitive\_ID = Auto-generate database key.

CDT\_SC\_ID = Foreign key to the DT\_SC.DT\_SC\_ID representing the target SC.

CDT\_Primitive\_ID = Foreign key to the CDT\_Primitive.CDT\_Primitive\_ID represents the CDT primitives in the Allowed Primitives column of the table in sections 4.X.8 of CCTS DTC3.

isDefault = This column indicates whether the associated CDT primitive is a default as documented in the CCTS DTC3. The values of this column are in sections 4.X.9.Y (Core Value Domains subsections). In each of the subsection, there is a table where the Allowed Primitive column either has only one allowed primitive or multiples. If there is only one allowed primitive, then that is the default, i.e., set the value to TRUE for that primitive (the rest of the allowed primitive in section 4.X.8 should be set as FALSE). If there are multiple, the table has another column indicating which primitive is the default.

##### Populate the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map table

This table is similar to the CDT\_Allowed\_Primitive\_Expression\_Type\_Map table in section (3.1.1.2.5), but it is for the SC primitive in the above section.

For each row in the CDT\_SC\_Allowed\_Primitive table, there will be zero or more rows in this table.

CDT\_SC\_Primitive\_Expression\_Type\_Map\_ID = Auto-generate database key.

CDT\_SC\_Allowed\_Primitive = Foreign key to CDT\_SC\_Allowed\_Primitive.CDT\_SC\_Allowed\_Primitive\_ID.

XSD\_BuiltIn\_Type\_ID = Foreign key to XSD\_BuiltIn\_Type. XSD\_BuiltIn\_Type\_ID

Use the CDT Primitive to XSD Built-in type in section 3.1.1.2.5 to populate the XSD\_BuiltIn\_Type\_ID column according to the SC allowed primitives.

#### Import Identifier Scheme

The content to be imported a merge of two schemas - <http://www.unece.org/uncefact/codelist/standard/UNECE_AgencyIdentificationCode_D13A.xsd> and IdentifierScheme\_AgencyIdentification\_3055\_D08B.xsd. This is because the code list schema uses some of the Agency ID that does not exist in version D08B. The merged schema should use the schema header, element names, and type names from the OAG schema but include all the ID values from the UNECE schema (a diff may be run to make sure of the changes needed to apply to the UNECE schema to match the OAG naming pattern).

##### Populate the Agency\_ID\_List table

There is only one record to be created here.

Agency\_ID\_List\_ID = Auto-generate database key.

Agency\_ID\_List\_GUID = //xsd:simpleType[endsWith(@name, “IdentificationContentType”)]/@id

Enumeration\_Type\_GUID = //xsd:simpleType[endsWith(@name, “EnumerationType”)]/@id

Name = “Agency Identification”.

List\_ID = “3055”

Agency\_ID = Foreign key to the Agency\_ID\_List\_Value table where the Value is “6”. This column will need to be populated after populating the Agency\_ID\_List\_Value table.

Version\_ID = “D08B”.

Definition = Copy the text from the top of the schema at <http://www.unece.org/uncefact/codelist/standard/UNECE_AgencyIdentificationCode_D08B.xsd>, where it says “Schema agency: UN/CEFACT Schema version: 3.3……”

##### Populate the Agency\_ID\_List\_Value table

The content shall be taken from the element //xsd:simpleType[endsWith(@name, “EnumerationType)]. Assume this is the context element for XPATH expressions below.

Agency\_ID\_List\_Value\_ID = Auto-generate database key.

Value = Move to descendant context elements xsd:enumearation. For each descendant xsd:enumeration element, populate this column with @value.

Name = Assuming still in the xsd:enumeration context element, populate this column with //ccts:Name.

Definition = Assuming still in the xsd:enumeration context element, populate this column with //ccts:Definition.

Owner\_Agency\_ID\_List\_ID = Foreign key to the Agency\_ID\_List\_ID populated in the previous section.

#### Import Code Lists

Import code list files in the *Code List Folder* at Model/Platform/2\_0/Common/CodeLists folder. Two tables are used in this import including Code\_List and Code\_Value. Code\_List stores the meta-data about the code list and the Code\_Value table stores all the code values.

##### Populate Code\_List table

Each schema in the Code List Folder has one or more xsd:simpleType with the following naming pattern <X>CodeContentType, <X> is the varying part. For example, CodeLists\_1.xsd has oacl\_ActionCodeContentType; and CodeList\_CharacterSetCode\_IANA\_20070514.xsd has clmIANACharacterSetCode20070514\_CharacterSetCodeContentType. Each of these must have a corresponding entry in the Code\_List table as described below. Note that XPATH expression below assumes that the context is that xsd:simpleType. For each entry in the Code\_List table, populate the Code\_List\_Value table as described in the next section.

Code\_List\_ID = Auto-generate database key.

Code\_List\_GUID = ./@id.

Enumeration\_Type\_GUID = If startsWith(<X>, “oacl”), the value is../xsd:simpleType[@name = concat(<X>, “EnumerationType”)]/@id. Otherwise, leave blank.0

Name = substring-before(/@name, ‘ContentType’). Ex. Name of the oacl\_ActionCodeContentType is “oacl\_ActionCode”.

List\_ID = Same as the Code\_List\_GUID for now.

Agency\_ID = This is a foreign key to the Agency\_ID\_List\_Value. Agency\_ID\_List\_Value\_ID associated with the Agency\_ID\_List\_Value. Value as follows. Code lists from CodeLists\_1.xsd, CodeList\_ConditionTypeCode\_1.xsd, CodeList\_ConstraintTypeCode\_1.xsd, CodeList\_DateFormatCode\_1.xsd, CodeList\_DateTimeFormatCode\_1.xsd, and CodeList\_TimeFormatCode\_1.xsd have the Value = “314” (taken from Scheme Agency Identification file). Code lists from CodeList\_CharacterSetCode\_IANA\_20070514.xsd and CodeList\_MIMEMediaTypeCode\_IANA\_7\_04.xsd have the Value = “379”. Codes from CodeList\_CurrentcyCode\_ISO\_7\_04.xsd and CodeList\_LanguageCode\_ISO\_7\_04.xsd have the value “5”. Codes from CodeList\_TimeZoneCode\_1.xsd have the Value = “5”. Codes from CodeList\_UnitCode\_UNECE\_7\_04.xsd have the Value = “6”.

Version\_ID = If the Name starts with ‘oacl’, the value is ‘1’. Else if the Name is ‘clm6Recommendation205\_MeasurementUnitCommonCode’, the value is ‘5’. Else the value is the number substring preceding the ‘\_’ in the name, e.g., if the Name is ‘clmIANAMIMEMediaType20090304\_MIMEMediaCode’, then the value is ‘20090304’.

Definition = Take it from xsd:annotation/xsd:documentation.

Definition\_Source = Take it from xsd:annotation/xsd:documentation/@source.

Based\_Code\_List\_ID = Empty.

Extensible\_Indicator = Default value is FALSE; however, if count(xsd:union) = 1, the value must be TRUE. In other words, the condition indicates that there is a union of the enumerated values and the xsd:token.

Created\_By\_User\_ID = “oagis”.

Last\_Updated\_By\_User\_ID = “oagis”.

Creation\_Timestamp = Current time.

Last\_Update\_Timestamp = Same as Creation\_Timestamp.

**Note:**

* We do not bring in the minLength and maxLength facets in this version because the current data model does not support it.
* External code list do not have a separate enumeration type like the OAG code list. At the (model) generation time, the Enumeration\_Type\_GUID column will be an indicator to whether an enumeration type need to be generated. This shouldn’t impact the standalone schema generation, i.e., the standalone schema should use the same pattern whether the code list is OAG’s or external.

##### Populate Code\_List\_Value table

The content of this table should be from xsd:simpleType[endsWith(@name, “CodeContentType”)] if count(xsd:simpleType[endsWith(@name, “CodeContentType”)]//xsd:enumeration) >= 1) or from xsd:simpleType[endsWith(@name, “EnumerationType”)] elements corresponding to the xsd:simpleType[endsWith(@name, “CodeContentType”)] in the previous section. The XPATH expressions below assume that the context element is one of the two xsd:simpleType.

Code\_List\_Value\_ID = Auto-generate database key.

Owner\_Code\_List\_ID = Foreign key to the corresponding Code\_List. Code\_List\_ID.

Value = Move context element to each descendant //xsd:enumeration element and populate this and the rest of the columns before moving to the next xsd:enumeration element. Populate this column with /@value.

Name = If the /@value is a valid dictionary word or set of words populate this column with the /@value. Otherwise, leave this blank.

Definition = .//xsd:documentation.

Definition\_Source = .//xsd:documentation/@source.

Used\_Indicator = True.

Locked\_Indicator = False.

#### Import default and unqualified BDTs from Fields.xsd

There are types defined in the Fields.xsd schema file corresponding to all CDTs *except* the Ordinal CDT (OAGIS 10 just didn’t implement the Ordinal CDT) (these types are under the “Data Types” schema comment line). For example, the Amount CDT has the AmountType defined. There are 19 unqualified BDTs corresponding to the 19 types in the schema file. These must be imported into the DT table as follows. Note that the XPATH expressions below assume the context element is xsd:complexType or xsd:simpleType of the BDT being captured.

For each of these unqualified BDTs, there will be two BDTs created – one for the unqualified BDT itself and another for the default BDT OAGIS adopted for each particular CDT. In the example snippet below, AmountType is the unqualified BDT and the AmountType\_0723C8 is the default BDT. Note that default BDTs are defined in the BusinessDataType\_1.xsd.

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

<xsd:simpleContent>

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

</xsd:simpleContent>

</xsd:complexType>

##### Populate BDTs in DT table.

DT\_ID = Auto-generate database key.

DT\_GUID = Get this from @id.

DT\_Type = “1” (note: 1 indicates BDT).

Version\_Number = “1.0”

Previous\_Version\_DT\_ID = Leave blank.

Revision\_Type = “0” (note: 0 means NEW).

Data\_Type\_Term = Same as that of the DT it is based on as indicated by the Based\_DT\_ID column.

Qualifier = Blank.

Based\_DT\_ID = Foreign key to the DT\_ID column of this table itself. This should point to the corresponding CDT for the default BDT and point to the default BDT for the unqualified BDT. For default BDT, identify the Based\_DT\_ID using the type name or documentation.

DEN = Take the type name remove the ‘Type’ substring and then concat with “. Type”.

Content\_Component\_DEN = First part of the DEN concat with ‘Content’, e.g., “Amount. Content”, “Amount\_0723C8. Content”.

Definition = Take the content from the /xsd:annotation/xsd:documentation/ccts:Definition for the default BDTs. Leave blank for the unqualified BDTs.

Content\_Component\_Definition = Take the content from //(xsd:extenstion or xsd:restriction or xsd:union)/xsd:annotation/xsd:documentation/ccts:Definition for the default BDTs. Leave blank for the unqualified BDTs.

Revision\_Documentation = Blank.

Revision\_State = “1” (note: 1 means published).

Created\_By\_User\_ID = “oagis”.

Last\_Updated\_By\_User\_ID = “oagis”.

Creation\_Timestamp = Current time.

Last\_Update\_Timestamp = Same as Creation\_Timestamp.

**Exceptions**

There are additional default BDTs and unqualified BDTs that need to be imported. These unqualified BDTs are the rests of the xsd:simpleType whose names do not end with the “CodeContentType” in the Fields.xsd. Before importing these unqualified BDTs, modify the Fields.xsd as described below, then import both the unqualified BDTs and default BDTs in the same way as described above.

Some modifications to Fields.xsd to do before the import.

* Change the DayDateType to restrict on DateType\_DB95C8
* Change the MonthDateType to restrict on DateType\_0C267D
* Change the MonthDayDateType to restrict on DateType\_5B057B
* Change the YearDateType to restrict on DateType\_57D5E1
* Change the YearMonthDateType to restrict on DateType\_BBCC14
* Modify text types. This will actually make the schema invalid (because the based types being changed to are complex type) but it should make the import code simpler because all these exceptions remain xsd:simpleType for distinguishing from other qualified BDTs to be imported next.
  + Change the NormalizedStringType to restrict on TextType\_0VCBX4
  + Change the TokenType to restrict on TextType\_0F0ZL2
  + Change the StringType to restrict on TextType\_62S0B4

Note that in the above statements “Change the X to restrict on Y”, X’s are in the unqualified BDTs category and Y’s are default BDTs category.

##### Populate BDT\_Primitive\_Restriction table

Assign CDT primitives and map XSD built-in types to the default BDTs and unqualified BDTs.

BDT\_Primitive\_Restriction\_ID = Auto-generate database key.

BDT\_ID = Foreign key to the BDTs populated in the previous section. There will be 1 or more rows in this table for each BDT because of the 1:m map between the CDT Primitive and XSD built-in type.

CDT\_Primitive\_Expression\_Type\_Map\_ID = This is a foreign key to the CDT\_Allowed\_Primitive\_Expression\_Type\_Map table in section 3.1.1.2.5. Each BDT, except those in the Exception subsection of the previous section, will use all the entries from that table per its associated CDT Primitive(s). Those in the Exception section should be bound to the row that matches its based XSD Built-in type (and CDT primitive) (add xsd:token for those dealing with time point, i.e., the DayDateType, MonthDayDateType, YearDateType, and YearMonthDateType). Each pair of default BDT and unqualified BDT has the same set of maps.

Code\_List\_ID = Leave blank.

isDefault = 1) For the default BDTs where there is a //xsd:union, isDefault should be set to TRUE on the xsd:token. 2) For default BDTs where is there is a //xsd:extension or //xsd:restriction, isDefault should be set to TRUE according to the XSD built-in type indicated in (//xsd:extension or //xsd:restriction)/@base. Use the same logic as in #2 for those BDTs in the Exception. All other rows are set to FALSE. For unqualified BDTs, inherit from their base.

**Note:**

1. In the future, we may associate xsd:token or xsd:string to all primitives so that the user can choose to extremely relax the data type constraint. This may be desired if the implementation does not want to do any XML schema level validation or that there is no more specific XSD built-in type that fit the requirement and the use of XSD union also does not work.

##### Populate SC in DT\_SC table

Populate the supplementary components for the default BDTs and unqualified BDTs.

For default BDTs, look for //xsd:attribute elements. Populate a row in this table for each //xsd:attribute of the default BDTs. For each unqualified BDT, inherit all SCs from its based default BDT, i.e., there must be the same number of rows as those SCs for its based default BDT.

DT\_SC\_ID = Auto-generate database key.

DT\_SC\_GUID = Take the value from //xsd:attribute/@id for the default BDTs. Inherit from the based default BDTs for the unqualified BDTs.

Property\_Term = The value is the same as that of the CDT on which the BDT is based, e.g., “Currency” for BDTs based on the Amount CDT. (Alternatively get this from the default BDTs at //xsd:attribute/xsd:annotation/xsd:documentation/ccts:PropertyTermName.)

Representation\_Term = The value is the same as that of the CDT on which the BDT is based, e.g., “Code” for the “Currency” supplementary component of BDTs based on the Amount CDT. (Alternatively get this from the default BDTs at //xsd:attribute/xsd:annotation/xsd:documentation/ccts:RepresentationTermName.)

Definition = Blank for the unqualified BDTs. For the default BDTs, get this from //xsd:attribute/xsd:annotation/xsd:documentation/ccts:Definition.

Owner\_DT\_ID = Foreign key to the corresponding BDTs table DT\_ID column populated in the previous section (3.1.1.5.1).

Min\_Cardinality = For the default BDTs, take the value from //xsd:attribute/@use. “optional” = 0. “required” = 1, “prohibited” = 0. If the attribute does not exist, it means 0. For the unqualified BDTs, the value is inherited from the based default BDT, unless the SC is a new attribute (extension) or the attribute is redefined again (i.e., count(//xsd:attribute) > = 1) then read the cardinality from the @use.

Max\_Cardinality = Set to 1 except for the NormalizedStringType, TokenType, and StringType. Set to zero for the languageCode attribute.

Based\_DT\_SC\_ID = For default BDTs, point to DT\_SC.DT\_SC\_ID of the corresponding CDT. For unqualified BDTs, point to the DT\_SC.DT\_SC\_ID of the default BDT on which it is based.

##### Populate BDT\_SC\_Primitive\_Restriction table

This table assigns the CDT primitive and map XSD built-in type combination to the default BDTs and unqualified BDTs.

BDT\_SC\_Primitive\_Restriction\_ID = Auto-generate database key.

BDT\_SC\_ID = Foreign key to the default BDT’s or the unqualified BDT’s SC.

CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map\_ID = This is a foreign key to the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map table in section 3.1.1.2.8. For default BDT, each BDT will use all the entries from that table per its associated CDT Primitive(s) except in cases where the SC’s property term or attribute name contains ‘Code’ and the type associated with the attribute is one of the code lists, i.e., assuming the context node is the xsd:attribute – /@type = Code\_List.Name.

If the type associated with the attribute is a code list, then first populate one row which have this column pointing to the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map\_ID that maps Code to Token and xsd:token. Then populate another row which has this column blank and but fill in the Code\_List\_ID column. Each unqualified BDT has the same set of maps as those of its based default BDT.

Code\_List\_ID = Populate this column with an appropriate Code\_List.Code\_List\_ID when the type associated with the attribute is determined to be a code list as described in the previous column.

Agency\_ID\_List\_ID = Leave blank.

isDefault = If the attribute is not bound to a code list, use the same default as in the BDT\_Primitive\_Restriction table according to the SC representation term. E.g., The Amount’s Currency Code SC representation term is Code. Look up the BDT\_Primitive\_Restriction.isDefault using a combination of Code, CDT Primitive, and XSD built-in type (according to the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map\_ID column) and apply the same TRUE OR FALSE value here. If the attribute is bound to a code list, set the isDefault to TRUE only on the record that the Code\_List\_ID column filled.

#### Import additional BDTs from Meta.xsd

There are three additional BDTs in Meta.xsd, namely ExpressionType, ActionExpressionType, ResponseExpressionType. The XPATH expression below assumes the context element is xsd:complexType of these types.

##### Populate the DT table

DT\_ID = Auto-generate database key.

DT\_GUID = Get it from /@id.

DT\_Type = “1” (note: 1 indicates BDT).

Version\_Number = “1.0”

Previous\_Version\_DT\_ID = Leave blank.

Revision\_Type = “0” (note: 0 means NEW).

Based\_DT\_ID = Based BDT of these types are the Text **default** BDT that has the xsd:token as the default primitive.

Data\_Type\_Term = Inherit from the based BDT identified in the Based\_DT\_ID column.

Qualifier = Leave blank.

DEN = SpaceSeparate(substring-before(/@name, “Type”)) + “. Type” and replace “ID” with “Identifier”.

Content\_Component\_DEN = DEN + “. Content”.

Definition = Use the value from ./xsd:annotation/xsd:documentation if any; otherwise leave empty.

Content\_Component\_Definition = Leave empty.

Revision\_Documentation = Leave empty.

Revision\_State = “1” (note: 1 means published).

Created\_By\_User\_ID = “oagis”.

Last\_Updated\_By\_User\_ID = “oagis”.

Creation\_Timestamp = Current time.

Last\_Update\_Timestamp = Same as Creation\_Timestamp.

**Note**:

1. Current version does not have a column to capture the schema module the BDT lives in. This will be done later.

##### Populate BDT\_Primitive\_Restriction table

Assign the record from the CDT\_Allowed\_Primitive\_Expression\_Type\_Map or Code\_List tables to the qualified BDTs based on its XSD built-in type or its associated code list.

|  |
| --- |
| BDT\_Primitive\_Restriction\_ID = Auto-generate database key. |
| BDT\_ID = Foreign key to the BDTs populated in the previous section. There will be 1 or more rows in this table for each BDT because of the 1:m map between the CDT Primitive and XSD built-in type. |
| CDT\_Primitive\_Expression\_Type\_Map\_ID = This is a foreign key to the CDT\_Allowed\_Primitive\_Expression\_Type\_Map table in section 3.1.1.2.5. Inherit all the maps for the Text default BDT. |
| Code\_List\_ID = Leave blank. |
| isDefault = Set the value to true for the row where CDT\_Primitive\_Expression\_Type\_Map\_ID points to xsd:token. |

##### Populate SC in the DT\_SC table

Populate the supplementary components for the BDTs. Populate a row for each BDT by inheriting all values from the “Language Code” SC of the based **default** Text BDT (“Language Code” is the Property\_Term of the SC) with two exceptions 1) set the Max\_Cardinality column to zero and 2) set the Based\_DT\_SC\_ID to the DT\_SC\_ID of the “Language Code” SC of the default Text BDT. Then populate a row in this table for each xsd:attribute declared in the xsd:complexType as described below.

DT\_SC\_ID = Auto-generate database key.

DT\_SC\_GUID = Take the value from .//xsd:attribute/@id.

Property\_Term = SpaceSeparate(substring-before(.//xsd:attribute/@name, ‘Code’)) and capitalize the first letter of the resulting string.

Representation\_Term = If .//xsd:attribute/@name = ‘expressionLanguage’, set to “Text”. If .//xsd:attribute/@name = ‘actionCode’ set to “Code”.

Definition = .//xsd:attribute/xsd:annotation/xsd:documentation.

Min\_Cardinality = Take the value from //xsd:attribute/@use. “optional” = 0. “required” = 1, “prohibited” = 0. If the @use attribute does not exist, it means 0.

Max\_Cardinality = Take the value from //xsd:attribute/@use. “optional” = 1. “required” = 1, “prohibited” = 0. If the @use attribute does not exist, it means 1.

Based\_DT\_SC\_ID = Leave blank.

##### Populate CDT\_SC\_Allowed\_Primitive table

Note that this section and the next section are needed because there is an attribute extension for each of these BDTs from Meta.xsd.

CDT\_SC\_Allowed\_Primitive\_ID = Auto-generate database key.

CDT\_SC\_ID = Foreign key to the DT\_SC.DT\_SC\_ID representing the target SC.

CDT\_Primitive\_ID = Foreign key to the CDT\_Primitive.CDT\_Primitive\_ID. For both ‘actionCode’ and ‘expressionLanguage’ attributes, this column should point to the NormalizedString, String, and Token CDT Primitive.

isDefault = The record that has the CDT\_Primitive\_ID column pointing to the Token CDT Primitive has this column value as TRUE. Other records have this column value as FALSE.

##### Populate CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map table

This table is similar to the CDT\_Allowed\_Primitive\_Expression\_Type\_Map table in section (3.1.1.2.5), but it is for the SC primitive in the above section.

For each row populated in the previous section (CDT\_SC\_Allowed\_Primitive) table, there will be zero or more rows in this table.

CDT\_SC\_Primitive\_Expression\_Type\_Map\_ID = Auto-generate database key.

CDT\_SC\_Allowed\_Primitive = Foreign key to CDT\_SC\_Allowed\_Primitive.CDT\_SC\_Allowed\_Primitive\_ID pointing to the each row populated in the previous section.

XSD\_BuiltIn\_Type\_ID = Foreign key to XSD\_BuiltIn\_Type. XSD\_BuiltIn\_Type\_ID

Use the CDT Primitive to XSD Built-in type in section 3.1.1.2.5 to populate the XSD\_BuiltIn\_Type\_ID column according to CDT primitive in the CDT\_SC\_Allowed\_Primitive column.

##### Populate the BDT\_SC\_Primitive\_Restriction table

This table assigns the CDT primitive and map XSD built-in type combination to the BDTs’ SCs.

|  |
| --- |
| BDT\_SC\_Primitive\_Restriction\_ID = Auto-generate database key. |
| BDT\_SC\_ID = Foreign key to the BDT’s SC. |
| For the “Language Code” inherited from the Text default BDT. Inherit all the maps (the CDT\_SC\_Primitive\_Expression\_Type\_Map\_ID’s, the Code\_List\_ID and the isDefault values) of the Text default BDT (the Language Code should have a map to an xsd:token and another map to the language Code list (clm56392A20081107\_LanguageCodeContentType).  For the expressionLanguage attribute, use the corresponding CDT\_SC\_Primitive\_Expression\_Type\_Map\_ID’s from the previous section and set the isDefault to true on the map to xsd:token..  For the actionCode attribute, populate rows for all the maps for the actionCode SC in the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map table with isDefault = FALSE. Populate another row with the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map\_ID column blank while the Code\_List\_ID column pointing to the “Action Code” code list and the isDefault column set to TRUE. |
| Agency\_ID\_List\_ID = Leave blank. |

#### Import BCCPs and Qualified BDTs

Basic Core Component Properties (BCCPs) are in Fields.xsd and Meta.xsd. BCCPs are xsd:element in those files that are eventually traced down to xsd:simpleContent through the type xsd:extension chain (or xsd:restriction, although I think xsd:restriction is not used at all) (note that all xsd:element in the Fields.xsd are BCCPs; however, not all xsd:element in the Meta.xsd are BCCPs so it is necessary to trace down to whether the xsd:element has a simple content or conversely does not have a complex content).

Assuming the target xsd:element is a BCCP, the xsd:element/@type is either an unqualified BDTs which we already imported in 3.1.1.5 or a new QBDT that needs to be created. The logic here is to check whether the BDT already exists in the database - if not create a new one; otherwise, associate with the existing one. To check whether a BDT already exists, we can use a GUID of the type (or name, but it is easier to use the GUID). In principle, multiple new BDTs may be necessary, i.e., a new QBDT may be basing on another new QBDT. Populate the BCCP table and DT related tables (for the new QBDT) as follows.

A QBDT should be defined in an xsd:complexType element (other than those xsd:complexType already imported as the default BDTs in 3.1.1.5) in the Fields.xsd. XPATH expressions below assume xsd:complexType is the context.

Assume there are following functions.

IdOf(object) returns the database key of the object. The database key is typically retrieved by matching with the DEN of the object.

GuidOf(object) returns GUID of an object.

DENxUUID(object) returns a DEN of the object without the UUID suffix at the end if any.

First(DEN) returns the first part of the given DEN before the “.” and ignore the ‘\_’. Ex. First(“Total\_ Amount. Type”) = “Total Amount”.

CamelCase(string) returns camel case representation of the given string.

SpaceSeparate(camelCaseString) returns space-separated format of the given camelCaseString.

<TableName>(<ID>).<ColumnName> returns the column value of the record identified by the <ID> in the <TableName> table and <ColumnName> column. Ex. BCCP(IdOf(./xsd:complexType/@name)).Definition means taking the value from the Definition column of the BCCP table whose record matches the database key returned by looking up the BCCP.DEN column.

##### Populate a QBDT in the DT table

DT\_ID = Auto-generate database key.

DT\_GUID = Get it from /@id.

DT\_Type = “1” (note: 1 indicates BDT).

Version\_Number = “1.0”

Previous\_Version\_DT\_ID = Leave blank.

Revision\_Type = “0” (note: 0 means NEW).

Based\_DT\_ID = Get from IdOf(SpaceSeparate(substring-before( //xsd:restriction/@base, ‘Type’))) or IdOf(SpaceSeparate(substring-before(//xsd:extension/@base) )) if the base is not a code content type, i.e., !endsWith(./@base, “CodeContentType”). If the base is a code content type, then this column should point to the CodeType default BDT. If neither of the IdOf function returns an ID, it means that the QBDT is based on another new QBDT. Cascade to create another QBDT and use the DT\_ID of that new QBDT. It is an exception for a QBDT, if a Based\_DT\_ID cannot be found.

Data\_Type\_Term = Inherit from the based BDT identified in the Based\_DT\_ID column.

Qualifier = SpaceSeparate(substring-before(/@name, “Type”)) – First(DENxUUID(Based\_DT\_ID)). If this results in a blank string, an exception or warning should be logged.

DEN = Qualifier + “\_ “ + DENxUUID(Based\_DT\_ID). Ex. DEN of OpenAmountType QBDT is “Open\_ Amount. Type”. It is an exception if this is empty.

Content\_Component\_DEN = substring-before(DEN, “. Type”) + “. Content”. Ex. “Open\_ Amount. Content”.

Definition = Use the value from ./xsd:annotation/xsd:documentation if any otherwise leave empty.

Content\_Component\_Definition = Leave empty.

Revision\_Documentation = Leave empty.

Revision\_State = “1” (note: 1 means published).

Created\_By\_User\_ID = “oagis”.

Last\_Updated\_By\_User\_ID = “oagis”.

Creation\_Timestamp = Current time.

Last\_Update\_Timestamp = Same as Creation\_Timestamp.

Some fixes to Fields.xsd to do before the import.

* Change the RecordFieldValueType to extend ValueType.

##### Populate BDT\_Primitive\_Restriction table

Assign the record from CDT\_Allowed\_Primitive\_Expression or Code\_List tables to the qualified BDTs based on its XSD built-in type or its associated code list.

|  |
| --- |
| BDT\_Primitive\_Restriction\_ID = Auto-generate database key. |
| BDT\_ID = Foreign key to the BDTs populated in the previous section. There will be 1 or more rows in this table for each BDT because of the 1:m map between the CDT Primitive and XSD built-in type. |
| CDT\_Primitive\_Expression\_Type\_Map\_ID = This is a foreign key to the CDT\_Allowed\_Primitive\_Expression\_Type\_Map table in section 3.1.1.2.5. All qualified BDTs inherit the same CDT\_Primitive\_Expression\_Type\_Map\_ID from its based DT except those whose Data\_Type\_Term is “Code”. Those with Code as its data type term inherits CDT\_Primitive\_Expression\_Type\_Map\_ID from its based DT if its //xsd:extension/@base = ‘CodeType’ (the Code unqualified BDT); otherwise, leave this column blank (the Code\_List\_ID column needs to be used instead). |
| Code\_List\_ID = Leave blank if the CDT\_Primitive\_Expression\_Type\_Map\_ID column is not empty; otherwise, this column needs to point to an appropriate record in the Code\_List table. If endsWith(//xsd:extension/@base, ‘CodeContentType’), retrieve an appropriate record using Code\_List.Name column by string contain condition, i.e., contains(Code\_List.Name, substring-before(.//xsd:extension/@base, ‘CodeContentType’)); else inherit the value from its based DT (i.e., in this case //xsd:extension/@base ends with “CodeType” which in turn relies on another code list as its primitive). |
| isDefault = Inherit the value from the based DT if the CDT\_Primitive\_Expression\_Type\_Map\_ID column is not empty; otherwise (Code\_List\_ID column is not empty), set the value to ‘True’. |

##### Populate SC in the DT\_SC table

Populate the supplementary components for the qualified BDTs.

Qualified BDTs inherit all SCs from its based BDT, populate rows in this table appropriately for these SCs by copying information from the SCs of the based BDT. In addition, a qualified BDT may extend the base with new attributes, i.e., count(.//xsd:attribute) > 0 (recall that the context element is xsd:complexType of the qualified BDT). Populate a row in this table for each new attribute.

DT\_SC\_ID = Auto-generate database key.

DT\_SC\_GUID = Take the value from xsd:attribute/@id for a new attribute. Inherit from the based BDT for copied over SC.

Property\_Term = Inherit from the based BDT for copied over SC. For the new SC, if the attribute name does not end with ‘Code’, ‘ID’, or ‘Value’, the property term is SpaceSeparate(xsd:attribute/@name) then capitalize the first letter of the resulting string (e.g., if the xsd:attribute/@name = “entryDateTime”, the property term is “Entry Date Time”). On the other hand, if the attribute name ends with ‘Code’, ‘ID’, or ‘Value’, the property term is substring-before(xsd:attribute/@name, ‘Code’ or ‘ID’ or ‘Value’) then capitalize the first letter of the resulting string (e.g., if the xsd:attribute/@name = “typeCode”, the property term is “Type”) – however, if substring-before(xsd:attribute/@name, ‘Code’ or ‘ID’ or ‘Value’) returns an empty string, use the whole xsd:attribute/@name and capitalize the first letter.

Representation\_Term = Inherit from the based BDT for copied over SC. For the new SC, the value depends on the xsd:attribute/@name or xsd:attribute/@type as follows. If the endsWith(xsd:attribute/@name, ‘Code’ or ‘code’) is true, the representation term is ‘Code’. If endsWith(xsd:attribute/@name, ‘Number’) is true, the representation term is ‘Number’. If endsWith(xsd:attribute/@name, ‘ID’) is true, the representation term is ‘Identifier’. If endsWith(xsd:attribute/@name, ‘DateTime’) is true, the representation term is ‘Date Time’. If endsWith(xsd:attribute/@name, ‘Value’) is true, the representation term is ‘Value’. If endsWith(xsd:attribute/@name, ‘name’) is true, the representation term is ‘Name’. For other cases, look at the xsd:attribute/@type. If xsd:attribute/@type = ‘StringType’ or ‘NormalizedStringType’, the representation term is ‘Text’. If xsd:attribute/@type = ‘IndicatorType’, the representation term is ‘Indicator’.

Definition = Blank for copied over SCs. For new SCs, get it from //xsd:attribute/xsd:annotation/xsd:documentation.

Owner\_DT\_ID = Foreign key to the corresponding qualified BDTs table DT\_ID column populated in the previous section (3.1.1.6.1).

Min\_Cardinality = Inherit from the based DT for copied over SCs. For new SCs, take the value from //xsd:attribute/@use. “optional” = 0. “required” = 1, “prohibited” = 0. If the @use attribute does not exist, it means 0.

Max\_Cardinality = Inherit from the based DT for copied over SCs. For new SCs, take the value from //xsd:attribute/@use. “optional” = 1. “required” = 1, “prohibited” = 0. If the @use attribute does not exist, it means 1.

Based\_DT\_SC\_ID = For copied over SCs, Based\_DT\_SC\_ID is the DT\_SC\_ID of the copied over record. For new SCs, this column is blank.

##### Populate CDT\_SC\_Allowed\_Primitive table

Only BDTs that have an attribute extension to its based BDT need to have records populated in this table and also the table in the next section.

CDT\_SC\_Allowed\_Primitive\_ID = Auto-generate database key.

CDT\_SC\_ID = Foreign key to the DT\_SC.DT\_SC\_ID representing the target SC.

CDT\_Primitive\_ID = Foreign key to the CDT\_Primitive.CDT\_Primitive\_ID. This column should be populated based on the SC’s representation term, DT\_SC.Representation\_Term. Use the representation term, to look up the necessary CDT\_Primitive\_ID’s from the CDT\_Allowed\_Primitive table. The look up is done by matching representation term with the data type term (DT.Data\_Type\_Term) after joining the DT table with the CDT\_Allowed\_Primitive table.

isDefault = Assign the same value obtained from the look up for the previous column.

##### Populate CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map table

This table is similar to the CDT\_Allowed\_Primitive\_Expression\_Type\_Map table in section (3.1.1.2.5), but it is for the SC primitive in the above section.

For each row in the CDT\_SC\_Allowed\_Primitive table, there will be zero or more rows in this table.

CDT\_SC\_Primitive\_Expression\_Type\_Map\_ID = Auto-generate database key.

CDT\_SC\_Allowed\_Primitive = Foreign key to CDT\_SC\_Allowed\_Primitive.CDT\_SC\_Allowed\_Primitive\_ID.

XSD\_BuiltIn\_Type\_ID = Foreign key to XSD\_BuiltIn\_Type. XSD\_BuiltIn\_Type\_ID

Use the CDT Primitive to XSD Built-in type in section 3.1.1.2.5 to populate the XSD\_BuiltIn\_Type\_ID column according to the SC allowed primitives.

##### Populate the BDT\_SC\_Primitive\_Restriction table

This table assigns the CDT primitive and map XSD built-in type combination to the QBDTs’ SCs.

This table needs to be populated for the SCs inherited from the based unqualified BDTs. For those SCs, all the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map\_ID and isDefault combinations and the Code\_List\_ID and isDefault combinations must be inherited.

|  |
| --- |
| BDT\_SC\_Primitive\_Restriction\_ID = Auto-generate database key. |
| BDT\_SC\_ID = Foreign key to the QBDT’s SC. |
| CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map\_ID and isDefault = This is a foreign key to the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map table in section 3.1.1.2.8. For SCs inherited from the based BDT, copy from its base (note that each QBDT’SC may have multiple entries/primitive-maps). For new SCs, use the following rules. If xsd:attribute/@type = ‘Number\_B98233’ is true, assign all Number CDT primitive map and set isDefault to true on the xsd:integer map. Else if xsd:attribute/@type = (‘CodeType\_1E7368’ or ‘CodeContentType’) or xsd:attribute/@name = ‘countryCode’, assign all Code CDT primitive map and set isDefault to true on the xsd:token. Else if xsd:attribute/@type = ‘StringType’, assign all Text CDT primitive map and set isDefault to true on the xsd:string. Else if xsd:attribute/@type = ‘NormalizedStringType’, assign all Text CDT primitive map and set isDefault to true on the xsd:string. Else if xsd:attribute/@name = ‘listID’ or ‘listVersionID’ or ‘unitCodeListVersionID’, assign all Identifier CDT primitive map and set isDefault to true on the xsd:normalizedString. Else if xsd:attribute/@type = ‘DateTimeType’, assign all DateTime CDT primitive map and set isDefault to true on the xsd:token. Else if xsd:attribute/@type = ‘IndicatorType’, assign all Indicator CDT primitive map and set isDefault to true on the xsd:boolean. Else if xsd:attribute/@type = ‘ValueType\_E7171E’, assign all Value CDT primitive map and set isDefault to true on the xsd:normalizedString. Else if xsd:attribute/@name = ‘name’, assign all Name CDT primitive map and set isDefault to true on the xsd:normalizedString. Else if contains(xsd:attribute/@type, ‘CodeContentType’) is true or xsd:attribute/@name = ‘listAgencyID’, assign all Code CDT primitive map and set isDefault to false on all these restrictions; populate an additional record using the Code\_List\_ID or Agency\_ID\_List\_ID column as described below (this additional record will have isDefault set to true). |
| Code\_List\_ID and isDefault: For SCs inherited from the based BDT, copy from its base. For new SCs, populate this column if the characteristics of the xsd:attribute fall thru the IF clauses described above and contains(xsd:attribute/@type, ‘CodeContentType’) is true. Assign the Code\_List.Code\_List\_ID from the record where contains(Code\_List.Name, substring-before(xsd:attribute/@type, ‘CodeContentType’) = true and set the isDefault to true. |
| Agency\_ID\_List\_ID and isDefault: For SCs inherited from the based BDT, copy from its base. For new SCs, populate this column if the characteristics of the xsd:attribute fall thru the IF clauses described above and xsd:attribute/@name = ‘listAgencyID’ is true. Assign the only Agency\_ID\_List.Agency\_ID\_List\_ID value there is in the Agency\_ID\_List table and set the isDefault to true. |

##### Populate the BCCP table

For each xsd:element which is a BCCP, a record must be created in the BCCP table. In XPATH expressions below, it is assumed that the context is xsd:element.

BCCP\_ID = Auto-generate database key.

BCCP\_GUID = Get from ./@id.

Property\_Term = Get from SpaceSeparate(/@name) and then replace the “ID” token with “Identifier”, e.g., “Discount Amount” for the DiscountAmount XSD element and “Accrued Amount” for the AccruedAmount XSD element.

Representation\_Term = Get from the DT.Data\_Type\_Term of the associated BDT. Ex. “Open\_ Amount” for the DiscountAmount XSD element, “Amount” for the AccruedAmount XSD element.

BDT\_ID = Foreign key to the DT.DT\_ID. Use this logic to find the DT.DT\_ID – IdOf(//xsd:simpleType[@name = ./@type] or //xsd:complexType[@name = ./@type]) where //xsd:simpleType and //xsd:complexType may locate in Fields.xsd or Meta.xsd. Use the @id attribute of the //xsd:simpleType or //xsd:complexType to match with the DT.DT\_GUID to get the DT.DT\_ID (alternatively DT.DEN may be used).

DEN = Property\_Term + “. “ + Representation\_Term. Truncate the ending part of the Property\_Term that overlaps with the beginning of the Representation\_Term. Ex. “Discount Amount. Open\_ Amount” for the DiscountAmount and “Accrued. Amount” for the AccruedAmount.

Definition = ./xsd:annotation/xsd:documentation.

Created\_By\_User\_ID = “oagis”.

Last\_Updated\_By\_User\_ID = “oagis”.

Creation\_Timestamp = Current time.

Last\_Update\_Timestamp = Same as Creation\_Timestamp.

#### Import ACCs, ASCCPs, BCCs and ASCCs

The condition xsd:complexType[count(xsd:simpleContent) = 0] is an indication that the xsd:complexType is an ACC. ASCCPs are global xsd:element which refers to an ACC as its type. Some ASCCPs are local xsd:element including the DataArea and Extension elements. There is a DataArea element for every BOD. ASCCs are descendant xsd:element of an ACC.

Look at the beginning of section 3.1.1.7 for non-xpath and non-java functions used in expressions below.

For each BOD file under the Model/BODs and Model/Platform/2\_0/BODs folders (e.g., AcknowledgeAllocateResource.xsd), start creating an ASCCP from the one and only one xsd:element in the file and drill down.

##### Populate the ASCCP table

Create a record in the ASCCP table for each xsd:element whose @type attribute refers to an xsd:complexType satisfying this condition xsd:complexType[count(xsd:simpleContent) = 0], i.e., the xsd:complexType that is an ACC. In addition, an ASCCP is created for the xsd:group definition as well (a group definition is the xsd:group[count(@name) = 1]). Note that XPATH expressions in this section assume that the context element is the xsd:element or xsd:group.

ASCCP\_ID = Auto-generate database key.

ASCCP\_GUID = Get from ./@id.

Property\_Term = SpaceSeparate(./@name).

Definition = ./xsd:annotation/xsd:documentation.

Role\_Of\_ACC\_ID = Foreign key IdOf(./@type) pointing to the ACC.ACC\_ID. The IdOf( ) should match the SpaceSeparate(substring-before(./@type, “Type”)) with the First(ACC.DEN) to get the ACC.ACC\_ID. If the IdOf( ) function does not find any ACC.ACC\_ID, that means the ACC has not been created (another option is to match the GUID instead by looking up the GUID of the type in the schema and match it with the one in the database). Cascade down to create the corresponding ACC first (find the xsd:complexType whose @name matches the xsd:element/@type and use that type to create the ACC record per 3.1.1.8.2 below). In the case of xsd:group, this is the foreign key IdOf([./@name](mailto:./@name)) pointing to the ACC.ACC\_ID representing the group, and typically the look up shouldn’t find it and that it needs to be created at the same time as its ASCCP counterpart.

DEN = Property\_Term + “. ” + First(ACC(Role\_Of\_ACC\_ID).DEN). For xsd:group, this is basically Property\_Term + “. ” + Property\_Term.

Created\_By\_User\_ID = “oagis”.

Last\_Updated\_By\_User\_ID = “oagis”.

Creation\_Timestamp = Current time.

Last\_Update\_Timestamp = Same as Creation\_Timestamp.

State = 4. **Note:** 4 means published.

Module = Name of the file where the type reside without the file extension, e.g., PurchaseOrder for an ASCCP from the PurchaseOrder.xsd.

Reusable\_Indicator = This is always true except the case when the ASCCP is a local element (i.e., count([./@ref](mailto:./@ref)) = 0).

##### Populate the ACC table

The XPATH expressions below assume that the context is xsd:complexType[count(xsd:simpleContent) = 0] or xsd:group[count(@name) = 1].

|  |
| --- |
| ACC\_ID = Auto-generate database key. |
| ACC\_GUID = Get from ./@id. |
| Object\_Class\_Term = SpaceSeparate(substring-before(./@name, ‘Type’)). |
| DEN = Object\_Class\_Term + “. Details”. |
| Definition = ./xsd:annotation/xsd:documentation. |
| Based\_ACC\_ID = Leave blank if the count(.//xsd:extension) = 0. Otherwise, the value is IdOf(./@base). Note the IdOf( ) function in this case should find the ACC.ACC\_ID of the row whose First(ACC.DEN) matches SpaceSeparate(substring-before(./@base, “Type”)). If there is no match, it means that the ACC has not been created. Therefore, cascade to create that ACC first. |
| Object\_Class\_Qualifier = Always empty. |
| OAGIS\_Component\_Type = If endsWith(First(DEN), “Base”), the value is 0. Else if [First(DEN) = “Open User Area” or “Any User Area” or “All Extension”] or endsWith(First(DEN), “Extension”), the value is 2. Else if endsWith(First(DEN), “Group”), the value is 3. Else the value is 1. **Note**: 0 = Base, 1 = Semantics, 2 = Extension, 3 = Semantic Group. |
| Created\_By\_User\_ID = “oagis”. |
| Last\_Updated\_By\_User\_ID = “oagis”. |
| Creation\_Timestamp = Current time. |
| Last\_Update\_Timestamp = Same as Creation\_Timestamp. |
| State = 4. **Note:** 4 means published. |
| Module = Name of the file where the type or the group reside without the file extension, e.g., Components for an ACC from the Components.xsd. |

After finish creating an ACC, prune through the ACC content (the xsd:complexType or xsd:group) to create necessary BCC, ASCCs, ASCCPs, and ACCs.

For each .//xsd:element[count(@ref) = 1] or .//xsd:group[count(@ref) = 1], look for an ASCCP or BCCP whose First(DEN) matches the SpaceSeparate(@ref), (alternatively, find by using the GUID). If it is an ASCCP then, a record must be created in the ASCC table (follow the logic in 3.1.1.8.3) (note that the xsd:group is an ASCCP). If it is a BCCP, then a record must be created in the BCC table (follow the logic in 3.1.1.8.4). If no match is found in either the ASCCP or BCCP table, the referred to entity should be an ASCCP (because the assumption is that the BCCPs have already been completely imported in the earlier section) and it has not been created. In this case, cascade to create the ASCCP first (follow the logic in 3.1.1.8.1) and then come back to create an ASCC record (follow the logic in 3.1.1.8.3).

If the .//xsd:element[count(@ref) = 0] (i.e., this is a local element), a new ASCCP has to be created (follow the logic in 3.1.1.8.1), then come back to create an ASCC (follow the logic in 3.1.1.8.3). Note: OAG uses only local-global (local element and global type) and does not use local-local (local element local type). Therefore, the latter is not accounted for here.

If count(.//xsd:attribute >=1 ), a BCC needs to be created for each of the attributes (follow the logic in 3.1.1.8.4).

Notice the dependency to other sections, this means that each of these sections should be written as a routine so that they can be recursively called.

##### Populate the ASCC table

XPATH expressions below assume that the context is an xsd:element.

ASCC\_ID = Auto-generate database key.

ASCC\_GUID = Get from ./@id.

Cardinality\_Min = If count(./@minOccurs = 0), the value is 1. Else the value is ./@minOccurs.

Cardinality\_Max = If count(./@maxOccurs = 0), the value is 1. Else if ./@maxOccurs = ‘unbounded’, the value is -1. Else the value is ./@maxOccurs.

Sequencing\_Key = This is an integer representing the position of the xsd:element under the xsd:sequence element of the xsd:complexType (the parent ACC). Note that the number starts at 1 even if the xsd:element is part of an extension (i.e., logically there is another preceding element).

Assoc\_From\_ACC\_ID = Foreign key pointing to the ACC.ACC\_ID that cause the creation of this ASCC.

Assoc\_To\_ASCCP\_ID = Foreign key to the ASCCP..ASCCP\_ID. The value is IdOf(./@ref) by matching the First(DEN) of the ASCCP, if the @ref attribute presents. If there is no @ref attribute, it is a local element and there should be an ASCCP record created right before this ASCC is being created (Look up by DEN is not possible in this case because of the duplicate names of local elements. The function to create an ASSCP should return the ASCCP\_ID of the record created. Alternatively, the ASCCP\_ID can be looked up by using the GUID).

DEN = First(ACC(Assoc\_From\_ACC\_ID).DEN) + “. “ + ASCCP(Assoc\_To\_ASCCP\_ID).DEN

Definition = ./xsd:annotation/xsd:document if exists.

##### Populate the BCC table

XPATH expressions below assume that the context is an xsd:element or xsd:attribute (descendants of an ACC xsd:complexType).

BCC\_ID = Auto-generate database key.

BCC\_GUID = Get from ./@id.

Cardinality\_Min = If the context is an xsd:element use the following rule. If count(./@minOccurs = 0), the value is 1. Else the value is ./@minOccurs. If the context is an xsd:attribute use this rule: Take the value from //xsd:attribute/@use. “optional” = 0. “required” = 1, “prohibited” = 0. If the @use attribute does not exist, it means 0.

Cardinality\_Max = If the context is an xsd:element use the following rule. If count(./@maxOccurs = 0), the value is 1. Else if ./@maxOccurs = ‘unbounded’, the value is -1. Else the value is ./@maxOccurs. If the context is an xsd:attribute use this rule: Take the value from //xsd:attribute/@use. “optional” = 1. “required” = 1, “prohibited” = 0. If the @use attribute does not exist, it means 1.

Assoc\_To\_BCCP\_ID = Foreign key to the BCCP.BCCP\_ID. The value is IdOf(./@ref) by matching the First(DEN). If there is no matching BCCP record, a new BCCP has to be created; however, this should only be the case when the BCC is an xsd:attribute. See 3.1.1.8.5 for how to populate these BCCPs.

Assoc\_From\_ACC\_ID = Foreign key pointing to the ACC.ACC\_ID causing the creation of this BCC.

Sequencing\_key = This is an integer representing the position of the xsd:element under the xsd:sequence element of the xsd:complexType (the parent ACC). Note that the number starts at 1 even if the xsd:element is part of an extension (i.e., logically there is another preceding element).

Entity\_Type = If the context is an xsd:element, the value is 1. Else the value is 0 (i.e., the context is an xsd:attribute).

DEN = First(ACC(Assoc\_To\_BCCP\_ID).DEN) + “. “ + BCCP(Assoc\_To\_BCCP\_ID).DEN

##### Populate the BCCP table for xsd:attribute

XPATH expressions below assume the context is an xsd:attribute.

BCCP\_ID = Auto-generate database key.

BCCP\_GUID = Generate a new GUID.

Property\_Term = Get from SpaceSeparate(./@name) and then replace the “ID” token with “Identifier”.

BDT\_ID = Foreign key to the DT.DT\_ID. IdOf(./@type). Use the First(DT.DEN) or GUID to match a DT record.

Representation\_Term = Get from the DT.Data\_Type\_Term of the associated BDT found in the previous column.

DEN = Property\_Term + “. “ + Representation\_Term

Definition = ./xsd:annotation/xsd:documentation, if exists.

Created\_By\_User\_ID = “oagis”.

Last\_Updated\_By\_User\_ID = “oagis”.

Creation\_Timestamp = Current time.

Last\_Update\_Timestamp = Same as Creation\_Timestamp.

#### Create ASCCPs for all Extension ACCs

This is a place holder. This can be done later when we want to export the OAGIS model.

### Verify the OAGIS 10 Model import

#### Checksum verification

The total number of each entity type in the database is checked against those obtained from running the XPATH query in the XML Spy. The table below summarizes the numbers.

Table : Quantity of each entity type

|  |  |  |
| --- | --- | --- |
| **Quantity** | **Description** | **XPATH Expression** |
| 3657 | Number of ACCs. Types in OAGIS.xsd and OAGIS-Nouns.xsd are excluded. | count(xsd:schema/xsd:complexType[count(xsd:simpleContent) = 0]) |
| 4268 | Number of ASCCP and BCCP. Elements in OAGIS.xsd and OAGIS-Nouns.xsd are excluded. Global elements in Extensions.xsd are excluded because they are never used. Local elements are included. Attributes cannot be included in the number (although they should be). | count(xsd:schema/xsd:element) + count(//xsd:element[count(@name) = 1 and name(..) != 'xsd:schema']) |
| 7164 | Number of ASCC and BCC. Exclude element references in OAGIS.xsd and OAGIS-Nouns.xsd. Local elements are included. Attributes are included. | count(//xsd:element[count(@ref) = 1]) + count(//xsd:element[count(@name) = 1 and name(..) != 'xsd:schema']) + count(//xsd:attribute[name(../..) != 'xsd:simpleContent']) |

In addition to checking quantity of each entity type, the GUIDs are also checked. The table below summarizes the XPATH query used to pull the GUIDs for each entity type.

|  |  |
| --- | --- |
| Description | XPATH Expression |
| ACCs. Types in OAGIS.xsd and OAGIS-Nouns.xsd are excluded. | xsd:schema/xsd:complexType[count(xsd:simpleContent) = 0]/@id |
| ASCCPs and BCCPs | xsd:schema/xsd:element/@id  //xsd:element[count(@name) = 1 and name(..) != 'xsd:schema']/@id |
| ASCCs and BCCs. | //xsd:element[count(@ref) = 1]/@id  //xsd:element[count(@name) = 1 and name(..) != 'xsd:schema']/@id  //xsd:attribute[name(../..) != 'xsd:simpleContent']/@id |

There are some mismatches between the number of GUIDs found and the number of entities found. This is because there are some entities with missing GUIDs.

1. There are 3 global elements with no GUIDs. Use this query to find those: xsd:schema/xsd:element[count(@id) = 0].
2. There are two attributes with no GUID. Use this query to find those: //xsd:attribute[name(../..) != 'xsd:simpleContent' and count(@id) = 0]
3. There are 18 element references with no GUID. Use this query to find those: //xsd:element[count(@ref) = 1 and count(@id) = 0]

#### Content verification

Verification will be done through the export of OAGIS content.

Exporting OAGIS model is not in scope at this point, so we will validate only with OAGIS standalone schemas.

A batch program needs to be developed to export all OAGIS standalone BODs and compare them with the existing standalone BODs in the OAGIS distribution.

#### Challenges

1. If there is an error, possible causes can come one or more of the three sources:
   1. The OAGIS import code
   2. The standalone export code
   3. The standalone schema generator code which generates the standalone schemas in the OAGIS distribution
2. The global element and type declarations may not be in the same order between the source (the exported standalone schema) and target (the standalone schema in the distribution). This difference is semantically indifferent.
3. The attribute declaration is also order insignificant.
4. Export logic for documentation may be different (more advance) than the documentation generated in the distribution schemas.

#### Methods to address the challenges

## Application Home Screen

Create a menu page of everything a user can do based on his/her role.

At this point the user should be able to access functionalities in 3.3 to 3.7.

For the names of the menus, we may want to make them configurable. The use case names in the following sections are default names. A deployment should map these use case names to menu names, e.g., Create a top-level ABIE = Create a standalone BOD.

## BIEs Management

### Create a top-level ABIE (a standalone BOD)

Select a top-level ACC from a list

Select a Business Context

Create a Business Context

Selected top-level ACC

Selected business context

Create a new business context

New business context

Top-level ABIE created

Create BIE records

##### Select a top-level ACC from a list

A list of top-level ACCs should be created for the user to select as a source for the top-level ABIE. A top-level ACC is a BOD in OAGIS.

There should be a condition on the ACC record in the configuration file which indicates how to find the top-level ACC. In the case of OAGIS, the condition is ‘Module != “Components” and Module != “Meta” and Module != “Noun” and Module != ‘Extensions’. The parameter name is TopLevelACCCondition. The reason to make this configurable is that this condition may change and a different condition may be needed for other OAGIS version or non-OAGIS library such as B2MML.

Make a query to select ACCs from the ACC table which meets the TopLevelACCCondition. This creates the top-level ACC list. The user selects an ACC from the list.

Validate to ensure that there is one and only one ASCCP associating to that ACC. Note that ASCCP is the BOD element and ACC is the BOD type.

##### Select a business context

A business context (BC) needs to be associated with the top-level ACC before a corresponding top-level ABIE can be created.

A UI should allow for the user to search for a business context and select one and return to this flow. See the 3.4.1 Search business context and 3.4.2 Select a business context.

##### Create BIE records

A corresponding BIE records are to be created in the database through a transaction set, but this set of transactions shouldn’t be committed yet (until the user saves or publishes this newly created top-level ABIE). Call the CreateBIEs function and give the top-level ACC as the argument.

###### Function: CreateTopLevelBIEs(tACC, bc)

Description: Create the top-level ASBIEP, the associated ABIE, and all descendant ABIEs given a top-level ACC and a business context.

Argument: tlACC = the given top-level ACC, bc = the given business context

Return: The top-level ASBIEP object.

CreateBIEs(tlACC, tlABIE)

tlASBIEP = CreateASBIEP(tlASCCP, tlABIE)

Return tlASBIEP

tlABIE = CreateABIE(tlACC, bc, true)

tlASCCP = QueryTopLevelASCCP(tlACC)

###### Function: CreateBIEs(gACC, gABIE)

Description: Create all descendant BIEs for the given ACC and its corresponding ABIE.

Argument: gACC = the given ACC, aABIE = aACC’s corresponding ABIE.

CreateAggregateDescendantBIEs(gACC, gABIE)

CreateBasicChildBIEs(gACC, gABIE)

###### Function: CreateABIE(gACC, bc, isTopLevel)

Description: Create an uncommitted ABIE record for the given ACC.

Argument: gACC = the given ACC, bc = Business Context, isTopLevel = indicating whether the ABIE is a top-level one

Return: An ABIE object

Populate the ABIE table as follows:

ABIE\_ID = Auto-generate database key.

ABIE\_GUID = Auto-generate GUID. Note see 3.10 Generate GUID about the GUID generation function.

Based\_ACC\_ID = gACC.ACC\_ID - foreign key to the gACC.

isTop\_Level = isTopLevel argument.

Business\_Context\_ID = Foreign key to the database key of the bc.

Definition = Null for now. The user will be able to specify later.

Created\_By\_User\_ID = Currently logged on user.

Last\_Updated\_By\_User\_ID = Currently logged on user.

Creation\_Timestamp = Current time. This should be the same for all BIEs being created under the same top-level ABIE.

Last\_Update\_Timestamp = Same as the Creation\_Timestamp.

State = “Saved”.

Client\_ID = Null. The user can specify later.

Version = Null. The user can specify later.

Status = Null. The user can specify later.

Remark = Null. The user can specify later.

Business\_Term = Null. The user can specify later.

###### Function: CreateASBIEP(gASCCP, tABIE)

Description: Create an uncommitted ASBIEP record given an ASCCP and its target ABIE.

Argument: gASCCP = the given ASCCP for which the ASBIP will be created, tABIE = the target ABIE to which the ASBIEP is associated

Return: An ASBIEP object

Populate the ASBIEP table as follows.

ASBIEP\_ID = Auto-generated database key.

ASBIEP\_GUID = Auto-generate GUID. Note see 3.10 Generate GUID about the GUID generation function.

Based\_ASCCP\_ID = Foreign key to gASCCP.ASCCP\_ID.

Role\_Of\_ABIE\_ID = tABIE.ABIE\_ID – foreign key to the tABIE.

Definition = Null. The user can specify later.

Remark = Null. The user can specify later.

Business\_Term = Null. The user can specify later.

Created\_By\_User\_ID = Currently logged on user.

Last\_Updated\_By\_User\_ID = Currently logged on user

Creation\_Timestamp = tABIE.Creation\_Timestamp.

Last\_Update\_Timestamp = Same as Creation\_Timestamp.

###### Function: CreateAggregateDescendantBIEs(gACC,gABIE)

Description: Create uncommitted records of direct child ASBIEs of the given ACC. Create the target ASBIEP of each ASBIE. Create the target ABIE of each ASBIEP and recursively call another function to create BIEs for the target ABIE.

Argument: gACC = the given ACC; gABIE = the corresponding ABIE of the gACC. The gABIE will be the owner (the association-from) of the child ASBIEs to be created.

ASCC[n] = QueryASCC(gACC)

For each ASCC[i], i = 1..n

i++ < n?

Y

aASBIE = CreateASBIE( ASCC[i], gABIE, aASBIEP)

aASCCP = QueryASCCP( ASCC[i])

tACC = QueryTargetACC( aASCCP)

aASBIEP = CreateASBIEP(aASCCP, tABIE)

CreateBIEs(tACC, gABIE.bc)

N

tABIE = CreateABIE(tACC, gABIE.bc, false)

Note: gABIE.bc is the business context of the gABIE.

###### Function: CreateBasicChildBIEs(gACC, gABIE)

Description: Create uncommitted records of direct child BBIEs of the given ACC. Create the target BBIEP of each BBIE.

Argument: gACC = the given ACC; gABIE = the corresponding ABIE of the gACC. The gABIE will be the owner (the association-from) of the child BBIEs to be created.

BCC[n] = QueryBCC(gACC)

For each BCC[i], i = 1..n

i++ < n?

Y

BBIE[i] = CreateBBIE(BCC[i], aABIE, BBIEP[i])

BCCP[i] = QueryBCCP(BCC[i])

BBIEP[i] = CreateBBIEP(BCCP[i])

N

###### Function: QueryTopLevelASCCP(tlACC)

Description: Get the ASCCP associated with the given top-level ACC

Argument: tlACC = the given top-level ACC

Return: An ASCCP object

There should be one and only one ASCCP for the given top-level ACC since it is a top-level ACC. Find the ASCCP in the database whose ASCCP.Role\_Of\_ACC\_ID match the tlACC.ACC\_ID.

###### Function: QueryASCC(gACC)

Description: Get all the ASCC children of the given ACC.

Argument: gACC = the given ACC

Return: An array of ASCC objects.

###### Function: QueryBCC(gACC)

Description: Get all BCC children of the given ACC.

Argument: gACC = the given ACC

Return: An array of the BCC objects.

###### Function: QueryBCCP(gBCC)

Description: Get the target BCCP of the given BCC.

Argument: gBCC = the given BCC.

Return: A BCCP object

###### Function: CreateASBIE(gASCC, pABIE, tASBIEP)

Description: Create an uncommitted ASBIE record for the given ASCC. The parent (associated-from ABIE) of the ASBIE is the given ABIE.

Argument: gASCC = the given ASCC, pABIE = the parent ABIE, tASBIEP = target ASBIEP

Return: An ASBIE object

Populate the ASBIE table as follows.

ASBIE\_ID = Auto-generated database key.

ASBIE\_GUID = Auto-generate GUID. Note see 3.10 Generate GUID about the GUID generation function.

Assoc\_From\_ABIE\_ID = Foreign key to the pABIE.ABIE\_ID.

Assoc\_To\_ASBIEP\_ID = Foreign key to the tASBIEP.ASBIEP\_ID.

Based\_ASCC = Foreign key to the ASCC.ASCC\_ID.

Definition = Null. The user can specify later.

Cardinality\_Min = Null. The user can change this later.

Cardinality\_Max = Null. The user can change this later.

Remark = Blank. The user can specify later.

Created\_By\_User\_ID = Currently logged on user.

Last\_Updated\_By\_User\_ID = Currently logged on user.

Creation\_Timestamp = Current time. This should be the same for all BIEs being created under the same top-level ABIE.

Last\_Update\_Timestamp = Same as the Creation\_Timestamp.

###### Function: CreateBBIE(gBCC, pABIE, tBBIEP)

Description: Create uncommitted BBIE record for the given BCC.

Argument: gBCC = the given BCC for which BBIE should be created, pABIE = parent ABIE of the gBCC, tBBIEP = target BBIEP to which the BBIE should associate.

Return: A BCC object

Populate the BBIE table as follows.

BBIE\_ID = Auto-generated database key.

BBIE\_GUID = Auto-generate GUID. Note see 3.10 Generate GUID about the GUID generation function.

Based\_BCC\_ID = Foreign key to the gBCC.BCC\_ID.

Cardinality\_Min = Inherit the value from gBCC.Cardinality\_Min.

Cardinality\_Max = Inherit the value from gBCC.Cardinality\_Max.

isNillable = 0.

Fixed\_Value = Null.

isNull = 0.

Assoc\_From\_ABIE\_ID = Foreign key to the pABIE.ABIE\_ID.

Assoc\_To\_BBIEP\_ID = Foreign key to the tBBIEP.BBIEP\_ID.

Definition = Null. The user can specify later.

Remark = Null. The user can specify later.

Created\_By\_User\_ID = Currently logged on user.

Last\_Updated\_By\_User\_ID = Currently logged on user.

Creation\_Timestamp = Current time. This should be the same for all BIEs being created under the same top-level ABIE.

Last\_Update\_Timestamp = Same as the Creation\_Timestamp.

Records for supplementary components of associated BBIEP (or thereof BCCP) have to be created in the BBIE\_SC table so that the user can customize the constraints later on. Get all the DT\_SC records used by the DT record associated with the underlying BCCP (note that the DT record is identified by the BCCP.BCCP\_BDT\_ID column).

For each DT\_SC record, populate BBIE\_SC table as follows.

BBIE\_SC\_ID = Auto-generated database key.

BBIE\_ID = Foreign key to the BBIE.BBIE\_ID generated above.

DT\_SC\_ID = Foreign key to the DT\_SC record, i.e., DT\_SC.DT\_SC\_ID.

Min\_Cardinality = Null. The user can specify later.

Max\_Cardinality = Null. The user can specify later.

DT\_SC\_Primitive\_Restriction\_ID = Null.

###### Function: CreateBBIEP(gBCCP)

Description: Create an uncommitted BBIEP record for the given BCCP

Argument: gBCCP = the given BCCP for the BBIEP should be created.

Return: BBIEP object

Populate the BBIEP table as follows.

BBIEP\_ID = Auto-generated database key.

BBIEP\_GUID = Auto-generate GUID. Note see 3.10 Generate GUID about the GUID generation function.

Based\_BCCP\_ID = Foreign key to the gBCCP.BCCP\_ID.

Definition = Null. The user can specify later.

Remark = Null. The user can specify later.

Business\_Term = Null. The user can specify later.

Created\_By\_User\_ID = Currently logged on user.

Last\_Updated\_By\_User\_ID = Currently logged on user.

Creation\_Timestamp = Current time. This should be the same for all BIEs being created under the same top-level ABIE.

Last\_Update\_Timestamp = Same as the Creation\_Timestamp.

#### Expand/Collapse a descendant ASBIE

#### Expand/Collapse a descendant BBIE

#### Customize a descendant ASBIE

#### Customize a descendant BBIE

### Save a top-level ABIE

### Edit a top-level ABIE

### Discard a top-level ABIE

### Publish a top-level ABIE

### Create a top-level ABIE by copy

## Business Context Management

Note that all use cases in the business context management includes the ‘Search business context use case’. In other words, all other use cases start from the search page.

A ‘Select’, ‘Create’, ‘Update’, ‘Delete’ button should be present on the BC Search UI.

Tables related to the BC management include Context\_Category, Context\_Scheme, Context\_Scheme\_Value, Business\_Context, and Business\_Context\_Value.

### Search business context

#### Bottom search

The UI should allow for searching of the BC based on Business\_Contex.Name and Context\_Scheme\_Value.Value, Context\_Scheme\_Value.Meaning.

#### Top search

The UI should allow for searching of the BC based on the Context\_Category.Name, Context\_Category.Description, and drill down until the user can select a Business\_Context record.

### Select a business context

The user selects a business context record and typically returns to the calling/depending use case. The use case returns the selected Business\_Context.ID or the whole business context object.

### Create a business context

The user specifies the name of the business context being created. Then the user can repeatedly specify a business context value using the ‘Add’ button. The user specifies the business context value by selecting a record from the Context\_Scheme\_Value table. See the ‘Create a Business Context’ screen in Figure 1. When the user clicks the ‘Add’ button in Figure 1, it brings up Figure 2 screen which helps the user search for the desired Context\_Scheme\_Value.



Figure : Create a Business Context screen

In the ‘Search and Select Context Value’ screen in Figure 2 the user searches for a Context\_Scheme\_Value by driving down from the Context\_Category search using its Context\_Category.Name or Context\_Category.Description. Multiple values can be selected before returning to the ‘Create a Business Context’ screen. Note that when presenting the Context\_Scheme for a Context\_Category search, the result should be ordered by Context\_Scheme.Scheme\_Agency\_ID, then by Scheme\_Name, then by Scheme\_Version.

After the user click the ‘Save’ button on the ‘Create a Business Context’ screen, appropriate records in the Business\_Context and Business\_Context\_Value tables should be created.



Figure : Search and Select Context Value screen

### Update a business context

## Context Category Management

### Create a context category

The user creates a Context Category business specifying the Name and Description corresponding to the columns in the Context\_Category table. The UI allows save and cancel actions.

## Context Scheme Management

The Context Scheme Management allows the user to manage the data in the Context\_Scheme and Context\_Scheme\_Value tables.

### Show the ‘Context Scheme Management Home’ page

This page should have the following menus – ‘Create a new context scheme’, ‘Manage context scheme value’.

### Create a new context scheme

The user specifies values corresponding to columns in the Context\_Scheme table. The column Scheme\_Name, Scheme\_Agency\_ID, Scheme\_Version, and Context\_Category\_ID are required. The Context\_Category\_ID should be a combo box showing Name and Description from the Context\_Category table.

The UI should have ‘Save & Exit’, and ‘Cancel’ buttons. The ‘Cancel’ button discards the creation. The ‘Save & Exit’ button creates a new record in the Context\_Scheme table. Before creating the record the following validation should be performed.

* All required fields are specified.
* If there is a record with the same Scheme\_Name, Scheme\_Agency\_ID, and Scheme\_Version in the table. If so, gives an error and go back to the screen for the user to make changes.
* If there is a record or more with the same Scheme\_Name and Scheme\_Agency\_ID but differing Scheme\_Version, then show a list of existing versions and ask the user to confirm that he/she wants to create the specified version. If the user does not confirm, then go back so the user can change the Scheme\_Version.

After cancellation or successful save & exit, return to the ‘Context Scheme Management Home’ page.

### Manage context scheme value

This page allows the user to manage data in the Context\_Scheme\_Value table. First, the user needs to choose the context scheme to work with. This maps to the Owner\_Context\_Scheme\_ID column. A search function that looks through the Context\_Scheme may be offered.

At this point, only adding a context value to the context scheme is covered. The UI should show existing context values the context scheme currently has in a table format that shows the Value and Meaning columns. The user cannot modify existing records. The table should allow the user to add more records by specifying at least the Value column.

The UI should have ‘Save & Exit’ and ‘Cancel’ buttons. The ‘Save & Exit’ button save addition records to the Context\_Scheme\_Value table and return to the ‘Context Scheme Management Home’ page. Before saving, check that the Value column is not empty and contains only valid characters [a-z], [A-Z], or [0-9]. The ‘Cancel’ discards all the added records and return to the ‘Context Scheme Management Home’ page.

## Generate an OAGIS Expression

### Generate a standalone XML Schema for a top-level ABIE

## Code List Management

### Create a new blank BIE code list

### Create a new BIE code list by restriction

### Save a working BIE code list

### Edit a BIE code list

## Manage CCs and DTs

### View CCs

## Generate GUID

Format of GUID should be configurable in a property file of the application. Maybe we can assume [prefix][delimiter][actual GUID].

# Logical View

## Overview

## Design Packages

# Process View

# Deployment View

# Implementation View

## Overview

## Layers

### Presentation Layer

### Control layer

### Resource Layer

### Domain Layer

### Common Layer

# Data View