Semantic Refinement Tool:

Software Design Document

Editor: Yunsu Lee, NIST/UMBC

Creation Date: 05/12/2014

Last Updated: 12/8/2015 9:06 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

Change Tracking

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

Table of Contents

[Change Tracking 2](#_Toc437328944)

[Table of Contents 3](#_Toc437328945)

[1. Introduction 8](#_Toc437328946)

[1.1 Purpose 8](#_Toc437328947)

[1.2 Scope 8](#_Toc437328948)

[1.3 Definitions, Acronyms and Abbreviations 8](#_Toc437328949)

[1.4 References 8](#_Toc437328950)

[2. Architectural Goals and Constraints 9](#_Toc437328951)

[2.1 Technical Platform 9](#_Toc437328952)

[2.2 Security 9](#_Toc437328953)

[2.3 Persistence 9](#_Toc437328954)

[2.4 Reliability/Availability 9](#_Toc437328955)

[2.5 Performance 9](#_Toc437328956)

[2.6 Internationalization 9](#_Toc437328957)

[3. Use-Case View 10](#_Toc437328958)

[3.1 Import OAGIS 10 Model into the Database 10](#_Toc437328959)

[3.1.1 Import OAGIS 10 Model into the Database 10](#_Toc437328960)

[3.1.1.1 Fix to OAGIS 10.1 10](#_Toc437328961)

[3.1.1.2 Create an OAGIS user 10](#_Toc437328962)

[3.1.1.3 Populate CDT data 10](#_Toc437328963)

[3.1.1.3.1 Populate the XSD\_BuiltIn\_Type table 10](#_Toc437328964)

[3.1.1.3.2 Populate the CDT\_Primitive table 11](#_Toc437328965)

[3.1.1.3.3 Populate CDTs in the DT table 11](#_Toc437328966)

[3.1.1.3.4 Populate the CDT\_Allowed\_Primitive table 12](#_Toc437328967)

[3.1.1.3.5 Populate the CDT\_Allowed\_Primitive\_Expression\_Type\_Map 12](#_Toc437328968)

[3.1.1.3.6 Populate CDTs’ supplementary component in the DT\_SC table 13](#_Toc437328969)

[3.1.1.3.7 Populate the CDT\_SC\_Allowed\_Primitive table 14](#_Toc437328970)

[3.1.1.3.8 Populate the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map table 14](#_Toc437328971)

[3.1.1.4 Import Identifier Scheme 14](#_Toc437328972)

[3.1.1.4.1 Populate the Agency\_ID\_List table 14](#_Toc437328973)

[3.1.1.4.2 Populate the Agency\_ID\_List\_Value table 15](#_Toc437328974)

[3.1.1.5 Import Code Lists 15](#_Toc437328975)

[3.1.1.5.1 Populate Code\_List table 15](#_Toc437328976)

[3.1.1.5.2 Populate Code\_List\_Value table 16](#_Toc437328977)

[3.1.1.6 Import default and unqualified BDTs from Fields.xsd 17](#_Toc437328978)

[3.1.1.6.1 Populate BDTs in DT table. 17](#_Toc437328979)

[3.1.1.6.2 Populate BDT\_Primitive\_Restriction table 18](#_Toc437328980)

[3.1.1.6.3 Populate SC in DT\_SC table 19](#_Toc437328981)

[3.1.1.6.4 Populate CDT\_SC\_Allowed\_Primitive table 20](#_Toc437328982)

[3.1.1.6.5 Populate CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map table 20](#_Toc437328983)

[3.1.1.6.6 Populate BDT\_SC\_Primitive\_Restriction table 20](#_Toc437328984)

[3.1.1.7 Import additional BDTs from Meta.xsd 21](#_Toc437328985)

[3.1.1.7.1 Populate the DT table 22](#_Toc437328986)

[3.1.1.7.2 Populate BDT\_Primitive\_Restriction table 22](#_Toc437328987)

[3.1.1.7.3 Populate SC in the DT\_SC table 22](#_Toc437328988)

[3.1.1.7.4 Populate CDT\_SC\_Allowed\_Primitive table 23](#_Toc437328989)

[3.1.1.7.5 Populate CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map table 23](#_Toc437328990)

[3.1.1.7.6 Populate the BDT\_SC\_Primitive\_Restriction table 24](#_Toc437328991)

[3.1.1.8 Pseudo functions used for describing the imports 24](#_Toc437328992)

[3.1.1.9 Import Qualified BDTs 25](#_Toc437328993)

[3.1.1.9.1 Populate a QBDT in the DT table 25](#_Toc437328994)

[3.1.1.9.2 Populate BDT\_Primitive\_Restriction table 26](#_Toc437328995)

[3.1.1.9.3 Populate SC in the DT\_SC table 26](#_Toc437328996)

[3.1.1.9.4 Populate CDT\_SC\_Allowed\_Primitive table 27](#_Toc437328997)

[3.1.1.9.5 Populate CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map table 27](#_Toc437328998)

[3.1.1.9.6 Populate the BDT\_SC\_Primitive\_Restriction table 28](#_Toc437328999)

[3.1.1.10 Import BCCPs 29](#_Toc437329000)

[3.1.1.10.1 Populate the BCCP table 29](#_Toc437329001)

[3.1.1.11 Import ACCs, ASCCPs, BCCs and ASCCs 30](#_Toc437329002)

[3.1.1.11.1 Populate the ASCCP table 30](#_Toc437329003)

[3.1.1.11.2 Populate the ACC table 30](#_Toc437329004)

[3.1.1.11.3 Populate the ASCC table 32](#_Toc437329005)

[3.1.1.11.4 Populate the BCC table 32](#_Toc437329006)

[3.1.1.11.5 Populate the BCCP table for xsd:attribute 33](#_Toc437329007)

[3.1.1.12 Create ASCCPs for all Extension ACCs 33](#_Toc437329008)

[3.1.1.13 Import other libraries 33](#_Toc437329009)

[3.1.2 Verify the OAGIS 10 Model import 33](#_Toc437329010)

[3.1.2.1 Checksum verification 33](#_Toc437329011)

[3.1.2.2 Content verification 34](#_Toc437329012)

[3.1.2.3 Challenges 34](#_Toc437329013)

[3.1.2.4 Methods to address the challenges 35](#_Toc437329014)

[3.2 Application Home Screen 35](#_Toc437329015)

[3.2.1 Context Scheme Management: Create a new context classification scheme, Search and select a context classification scheme 35](#_Toc437329016)

[3.3 BIEs Management 36](#_Toc437329017)

[3.3.1 Create a top-level ABIE (a standalone BOD) 36](#_Toc437329018)

[3.3.1.1.1 Select a top-level concept 36](#_Toc437329019)

[3.3.1.1.2 Select a business context 36](#_Toc437329020)

[3.3.1.1.3 Create uncommitted BIE records/objects 37](#_Toc437329021)

[3.3.2 Edit a top-level ABIE 44](#_Toc437329022)

[3.3.3 Show top-level ABIE 44](#_Toc437329023)

[3.3.4 Save a top-level ABIE 46](#_Toc437329024)

[3.3.5 Discard a top-level ABIE 46](#_Toc437329025)

[3.3.6 Publish a top-level ABIE 46](#_Toc437329026)

[3.3.7 Top-level ABIE search and select 46](#_Toc437329027)

[3.3.8 Copy a top-level ABIE 46](#_Toc437329028)

[3.3.9 Extend a top-level ABIE 46](#_Toc437329029)

[3.3.9.1 Create a New User Extension Group ACC 49](#_Toc437329030)

[3.3.9.1.1 Populate an ACC current record 49](#_Toc437329031)

[3.3.9.1.2 Populate an ASCCP current record 50](#_Toc437329032)

[3.3.9.1.3 Populate an ASCC current record 50](#_Toc437329033)

[3.3.9.1.4 Populate an ACC history Record 51](#_Toc437329034)

[3.3.9.1.5 Populate an ASCCP history record 51](#_Toc437329035)

[3.3.9.1.6 Populate an ASCC history record 51](#_Toc437329036)

[3.3.9.2 Create a New User Extension Group ACC Revision 52](#_Toc437329037)

[3.3.9.3 Edit User Extension Group ACC 52](#_Toc437329038)

[3.3.10 The UI should show a tree model of the Model BOD (CC) correspondnig to the tlABIE. All nodes except the Extension element corresponding to the eACC should be uneditable. Uneditable nodes may be rendered in a different background. In Figure 3, for example, uneditable nodes are rendered with the blue background. Note that since this is a CC rendering the ACC whose component type is 3 (Semantic Group) will be rendered; so as the type extension. See 3.10.1 for basic Architecture for managing revisions 52](#_Toc437329039)

[3.3.10.1 Current and history records 52](#_Toc437329040)

[3.3.10.2 CC states 53](#_Toc437329041)

[3.3.10.3 Unit of control and concurrency management 54](#_Toc437329042)

[3.3.10.4 Release management 55](#_Toc437329043)

[3.3.11 CCs and DTs management home screen 55](#_Toc437329044)

[3.3.11.1 Review User Extension Group ACC 57](#_Toc437329045)

[3.3.11.2 Append a child BBIE 57](#_Toc437329046)

[3.3.11.2.1 Reuse a BCCP 57](#_Toc437329047)

[3.3.11.2.2 Create a new BCCP 58](#_Toc437329048)

[3.3.11.2.3 Displaying BDT 58](#_Toc437329049)

[3.3.11.3 Append a child ASBIE 59](#_Toc437329050)

[3.3.11.3.1 Reuse an ASCCP 59](#_Toc437329051)

[3.3.11.4 Delete a child BBIE or ASBIE 59](#_Toc437329052)

[3.3.11.5 Move a child BBIE or ASBIE (currently out of scope) 59](#_Toc437329053)

[3.4 Business Context Management 59](#_Toc437329054)

[3.4.1 Search and select a business context 59](#_Toc437329055)

[3.4.1.1 Bottom search 59](#_Toc437329056)

[3.4.1.2 Top search 60](#_Toc437329057)

[3.4.2 Create a business context 60](#_Toc437329058)

[3.4.3 Show a business context 61](#_Toc437329059)

[3.4.4 Update a business context 62](#_Toc437329060)

[3.4.5 Discard a business context 62](#_Toc437329061)

[3.5 Context Category Management 62](#_Toc437329062)

[3.5.1 Create a context category 62](#_Toc437329063)

[3.5.2 Search and select a context category 62](#_Toc437329064)

[3.5.3 Update a context category 62](#_Toc437329065)

[3.5.4 Show a context category 62](#_Toc437329066)

[3.5.5 Discard a context category 63](#_Toc437329067)

[3.6 Context Classification Scheme Management 63](#_Toc437329068)

[3.6.1 Create a new context classification scheme 63](#_Toc437329069)

[3.6.2 Search and select a context classification scheme 64](#_Toc437329070)

[3.6.3 Update a context classification scheme 64](#_Toc437329071)

[3.6.4 Discard a context classification scheme 65](#_Toc437329072)

[3.7 Code List Management 65](#_Toc437329073)

[3.7.1 Create a new BIE code list without a base 65](#_Toc437329074)

[3.7.2 Create a new BIE code list from another (by extension & restriction) 66](#_Toc437329075)

[3.7.3 Search and select a code list 67](#_Toc437329076)

[3.7.4 View/Edit a BIE code list 67](#_Toc437329077)

[3.7.5 Discard a BIE code list 67](#_Toc437329078)

[3.7.6 Delete a BIE code list 68](#_Toc437329079)

[3.8 Generate a standalone OAGIS Expression for a top-level ABIE 68](#_Toc437329080)

[3.8.1 Generate a standalone XML Schema 69](#_Toc437329081)

[3.8.1.1 GenerateTopLevelABIE(tlASBIEP, tlABIEDOM) 69](#_Toc437329082)

[3.8.1.2 GenerateTopLevelASBIEP(gTlASBIEP, gSchemaNode) 70](#_Toc437329083)

[3.8.1.3 GenerateTopLevelASBIEPAnno(gTlASBIEP, gRootENode) 70](#_Toc437329084)

[3.8.1.4 IsAnnoOptionEmpty( ) 71](#_Toc437329085)

[3.8.1.5 GenerateABIE(gABIE, gASBIEPElementNode) 71](#_Toc437329086)

[3.8.1.6 GenerateABIEsAnno(gABIE, gABIECTypeNode) 71](#_Toc437329087)

[3.8.1.7 GenerateBIEs(gABIE, gPNode, gSchemaNode) 71](#_Toc437329088)

[3.8.1.8 QueryTargetABIE(gASBIEP) 72](#_Toc437329089)

[3.8.1.9 QueryChildBIEs(gABIE) 72](#_Toc437329090)

[3.8.1.10 GenerateASBIE(gASBIE, gPNode) 73](#_Toc437329091)

[3.8.1.11 QueryAssocToASBIEP(gASBIE) 73](#_Toc437329092)

[3.8.1.12 GenerateASBIEP(gASBIEP, gElementNode) 73](#_Toc437329093)

[3.8.1.13 QueryTargetABIE(gASBIEP) 73](#_Toc437329094)

[3.8.1.14 QueryAssocBDT(gBBIE) 74](#_Toc437329095)

[3.8.1.15 GenerateBBIE(gBBIE, gBDT, gPNode, gSchemaNode) 74](#_Toc437329096)

[3.8.1.16 GetBBIEElementName(gBBIE) 76](#_Toc437329097)

[3.8.1.17 GetBBIEAttributeName(gBBIE) 76](#_Toc437329098)

[3.8.1.18 QueryBBIESCs(gBBIE) 77](#_Toc437329099)

[3.8.1.19 isCodeListGenerated(gCL) 77](#_Toc437329100)

[3.8.1.20 GetCodeListTypeName(gCL) 77](#_Toc437329101)

[3.8.1.21 GenerateCodeList(gCL, gBDT | gSC, gSchemaNode) 77](#_Toc437329102)

[3.8.1.22 GenerateSCs(gBBIE, gBBIENode, gSCs[ ], gSchemaNode) 78](#_Toc437329103)

[3.8.1.23 isAgencyListGenerated(gAL) 80](#_Toc437329104)

[3.8.1.24 GetAgencyListTypeName(gAL) 80](#_Toc437329105)

[3.8.1.25 GenerateAgencyList(gAL, gSC, gSchemaNode) 80](#_Toc437329106)

[3.9 Namespace Management 81](#_Toc437329107)

[3.10 Manage CCs and DTs 81](#_Toc437329108)

[3.10.1 Architecture for managing revisions 81](#_Toc437329109)

[3.10.1.1 Current and history records 81](#_Toc437329110)

[3.10.1.2 CC states 82](#_Toc437329111)

[3.10.1.3 Summary of state related business rules 84](#_Toc437329112)

[3.10.1.4 Unit of control and concurrency management 84](#_Toc437329113)

[3.10.1.5 Release management 85](#_Toc437329114)

[3.10.2 CCs and DTs management home screen 85](#_Toc437329115)

[3.10.3 ACC and ASCCP content rendering 85](#_Toc437329116)

[3.10.4 View CCs 86](#_Toc437329117)

[3.10.5 Create a new ACC (and an ASCCP) 86](#_Toc437329118)

[3.10.5.1 ACC current record 87](#_Toc437329119)

[3.10.5.2 ACC history record 88](#_Toc437329120)

[3.10.5.3 ASCCP current record 88](#_Toc437329121)

[3.10.5.4 ASCCP history record 88](#_Toc437329122)

[3.10.6 Create a new revision of an ACC 88](#_Toc437329123)

[3.10.6.1 Notification 88](#_Toc437329124)

[3.10.7 Edit an ACC 88](#_Toc437329125)

[3.10.8 Add a child association to an ACC 88](#_Toc437329126)

[3.10.8.1 Populate the ASCC current record 89](#_Toc437329127)

[3.10.8.2 Popultate the ASCC history record 89](#_Toc437329128)

[3.10.8.3 Populate the BCC current record 90](#_Toc437329129)

[3.10.8.4 Populate the BCC history record 90](#_Toc437329130)

[3.10.9 Insert a child association to an ACC 90](#_Toc437329131)

[3.10.10 Delete a child association from an ACC 90](#_Toc437329132)

[3.10.11 Edit/Revise an association 91](#_Toc437329133)

[3.10.12 Make an ACC based on another ACC 91](#_Toc437329134)

[3.10.13 Create a new ASCCP 91](#_Toc437329135)

[3.10.14 Create a new revision of an ASCCP 91](#_Toc437329136)

[3.10.15 Create a new BCCP 91](#_Toc437329137)

[3.10.16 Create a new revision of a BCCP 91](#_Toc437329138)

[3.10.17 Deprecation 91](#_Toc437329139)

[3.10.18 Undoing 91](#_Toc437329140)

[3.10.19 Event notification 91](#_Toc437329141)

[3.11 Generate GUID 91](#_Toc437329142)

[4. Configuration File 92](#_Toc437329143)

[4.1 SQL for Top-level Concept 92](#_Toc437329144)

[4.2 Expression type mapping display configuration 92](#_Toc437329145)

[4.3 UI related variables 92](#_Toc437329146)

[5. Logical View 94](#_Toc437329147)

[5.1 Overview 94](#_Toc437329148)

[5.2 Design Packages 94](#_Toc437329149)

[6. Process View 95](#_Toc437329150)

[7. Deployment View 96](#_Toc437329151)

[8. Implementation View 97](#_Toc437329152)

[8.1 Overview 97](#_Toc437329153)

[8.2 Layers 97](#_Toc437329154)

[8.2.1 Presentation Layer 97](#_Toc437329155)

[8.2.2 Control layer 97](#_Toc437329156)

[8.2.3 Resource Layer 97](#_Toc437329157)

[8.2.4 Domain Layer 97](#_Toc437329158)

[8.2.5 Common Layer 97](#_Toc437329159)

[9. Data View 98](#_Toc437329160)

# 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

#### Fix to OAGIS 10.1

There are some fixes that needs to be done to OAGIS 10.1 before executing the imports

1. The StartTime and EndTime elements in Fields.xsd use xbt\_DayOfWeekHourMinuteUTCType, which is not a BDT. To fix, create another qualified BDT derived from the TimeType\_100CDA to reflect the name DayOfWeekHourMinuteUTCType. Note that the TimeType\_100CDA will be bound to xsd:token based on the BDT import logic below. Copy the documentation from the xbt\_DayOfWeekHourMinuteUTCType to the new BDT. Then use the new BDT with the two elements.
2. There are some local elements, fix them to correctly use the corresponding global elements.

#### 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 |
| 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 |

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 (3.1.1.3.5), 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 3.1.1.3.5 to populate the XSD\_BuiltIn\_Type\_ID column according to the SC allowed primitives.

#### Import Identifier Scheme

The content to be imported is 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.

State = “Published”

**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.

Extension\_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.
  + Add the following types to the BusinessDataType\_1.xsd.
    - TextType\_62S0C1 as an xsd:simpleType with restriction on xsd:string.
    - TextType\_0VCBZ5 as an xsd:simpleType with restriction on xsd:normalizedString.
    - TextType\_0F0ZX1 as an xsd:simpleType with restriction on xsd:token.
  + Change the NormalizedStringType to restrict on TextType\_0VCBZ5
  + Change the TokenType to restrict on TextType\_0F0ZX1
  + Change the StringType to restrict on TextType\_62S0C1

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.

###### Additional default BDTs

There are additional default BDTs used directly in the xsd:attribute within the Meta.xsd, Fields.xsd, Components.xsd, and Noun files (both directly under the Platform and under the Model folders). The way to retrieve these types is to run an XPATH query across these files and check one by one whether it has already been imported. The XPATH query to use is distinct-values(//xsd:attribute/@type). Check by comparing DEN to whether the type has already been imported (note that for the type whose name ends with ‘CodeContentType’, its first part of DEN is SpaceSeparate(substring-before(@type))). If not, use the logic above to populate the DT table and the rest of the sections below to populate other tables. Note that these are considered default BDTs, so follow the description for default BDTs.

Additionally, import the ValueType\_039C44 in the BusinessDataType\_1.xsd. This type is used in the PrioritySequenceValue BCCP in the Fields.xsd.

##### 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 3.1.1.3.5. 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 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 (some drill downs may be necessary until seeing the base which is an XSD built-in type). 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 the default BDTs, there must be a row in this table for each SC of its based CDT. For each unqualified BDT, copy all SCs from its based default BDT (except the ID columns), then look for an additional SC in its .//xsd:attribute.

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 = If the default BDTs have the xsd:attribute corresponding to the SC, take the value from //xsd:attribute/@id. If not generate a new one. For the unqualified BDT, generate a new GUID for inherited attribute. Use the existing //xsd:attribute/@id for the new SC added to the unqualified BDT.

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.) For a new SC in the unqualified BDT, use the spaceSeparate(xsd:attribute/@name).

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.) For the new SC in the unqualified BDT, use the last token of the spaceSeparate(xsd:attribute/@name), e.g., if name is “sequenceNumber”, the Representation\_Term is “Number”.

Definition = Get this from //xsd:attribute/xsd:annotation/xsd:documentation/ccts:Definition. If that does not exist or blank, get it from the base.

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

Min\_Cardinality = For the default BDTs, take the value from //xsd:attribute/@use. “optional” = 0. “required” = 1, “prohibited” = 0. If the //xsd:attribute/@use does not exist, it means 0. For the SC that has no corresponding xsd:attribute, the value is 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 CDT\_SC\_Allowed\_Primitive table

Only unqualified BDTs that have an attribute extension to its based default BDT need to have records populated in this table and also the table in the next section. In OAGIS 10.1, there is only one unqualified BDT that has an additional SC to its based default BDT – the NameType.

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 (3.1.1.3.5), 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 3.1.1.3.5 to populate the XSD\_BuiltIn\_Type\_ID column according to the SC allowed primitives.

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

This table assigns the CDT primitive and map XSD built-in type combination to the SC of 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 3.1.1.3.8. For default BDT, each BDT SC will use all the entries from that table per its associated CDT Primitive(s) except in cases where 1) 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; and 2) the attribute name contains ‘AgencyID’ and the type associated with the attribute is an agency ID list.

If the type associated with the attribute is a code list, then first populate one row which has this column pointing to the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map\_ID that maps the SC to Token and xsd:token. Then populate another row which has this column blank but fill in the Code\_List\_ID column.

If attribute name contains ‘AgencyID’, then first populate one row which has this column pointing to the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map\_ID that maps the SC to Token and xsd:token. Then populate another row which has this column blank but fill in the Agency\_ID\_List\_ID column with the only Agency\_ID\_List.Agency\_ID\_List\_ID in the database.

Each unqualified BDT has the same set of maps as those of its based default BDT. For a new SC in the unqualified BDT, first populate all rows from the map table for that SC (i.e., the value in the BDT\_SC\_ID column above matches the CDT\_SC\_Allowed\_Primitive.CDT\_SC\_ID column after joining the CDT\_SC\_Allowed\_Primitive and the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map tables). Then use the following rules to set the isDefault column. If xsd:attribute/@type = ‘Number\_B98233’ is true, set isDefault to true on the row, which points to the Number CDT primitive and xsd:integer map.

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.

If the type

Agency\_ID\_List\_ID = See the logic associated with the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map\_ID column above.

isDefault = Follow the table below for setting the isDefault value.

|  |  |
| --- | --- |
| SC Representation Term | The map [CDT\_Primitive, XSD Builtin Type] derived from the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map\_ID |
| Code | True on [Token, xsd:token], false on all other rows (including code list) |
| Identifier | True on [Token, xsd:token], false on all other rows |
| Name | True on [Token, xsd:token], false on all other rows |
| Indicator | True on [Boolean, xsd:boolean], false on all other rows |
| Value | True on [Decimal, xsd:decimal], false on all other rows |
| Text | True on [Text, xsd:string], false on all other rows |
| Number | True on [Decimal, xsd:decimal], false on all other rows |
| Date Time | True on [Timepoint, xsd:token], false on all other rows |

#### 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 3.1.1.3.5. 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 (3.1.1.3.5), 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 3.1.1.3.5 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. |

#### Pseudo functions used for describing the imports

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”.

UpperCamelCase(string) returns the upper camel case representation of the given string. It removes space and the underscore character. It also abbreviates ‘Identifier’ to ‘ID’. Ex. UpperCamelCase(“Type Code”) = “TypeCode”, UpperCamelCase(“List Identifier”) = “ListID”.

LowerCamelCase(string) returns the lower camel case representation of the given string. Everything is the same as the UpperCamelCase function, except that the starting character should be a lower case. Ex. LowerCamelCase(“Type Code”) = “typeCode”, LowerCamelCase(“List Identifier”) = “listID”.

SpaceSeparate(camelCaseString) returns space-separated format of the given camelCaseString. If the first letter of the string is lower case, capitalize it. Ex. SpaceSeparate(“typeCode”) = “Type Code”. SpaceSeparate(“LogicalID”) = “Logical ID”.

<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.

#### Import Qualified BDTs

Import the QBDTs from the Fields.xsd. Run through all the types declaration in the Fields.xsd, check with the GUID whether the type has already been imported in the database. If not, check whether it has a simple content. The type has a simple content if it meets this XPATH condition – xsd:complexType[count(.//xsd:simpleContent >= 1]. The type that passes these tests shall be imported as a qualified BDTs. (Note that this logic assumes that all simple types have already been imported as other types of DT earlier. In other words, there is no QBDT declared as a simpleType)

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 3.1.1.6) in the Fields.xsd. So this DT import needs to happen after all earlier DT imports.

XPATH expressions below assume xsd:complexType is the context.

##### 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 unqualified 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 3.1.1.3.5. 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 may in turn rely 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’~~ ‘False’. |

##### 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 except when the attribute represents the same SC as the one in the based BDT (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. Generate a new GUID for the unqualified BDTs 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 = Inherit from base 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 (3.1.1.3.5), 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 3.1.1.3.5 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.

|  |
| --- |
| 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 3.1.1.3.8. For SCs inherited from the based BDT, copy from its base or ancestor (note that each QBDT’SC may have multiple entries/primitive-maps). For new SCs, first populate all rows from the map table for that SC (i.e., the value in the BDT\_SC\_ID column above matches the CDT\_SC\_Allowed\_Primitive.CDT\_SC\_ID column after joining the CDT\_SC\_Allowed\_Primitive and the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map tables). Then use the following rules to set the isDefault column. If xsd:attribute/@type = ‘Number\_B98233’ is true, set isDefault to true on the row, which points to the Number CDT primitive and xsd:integer map. Else if xsd:attribute/@type = (‘CodeType\_1E7368’ or ‘CodeContentType’) or xsd:attribute/@name = ‘countryCode’, set isDefault to true on the row, which points to the Token CDT primitive and xsd:token map. Else if xsd:attribute/@type = ‘StringType’, set isDefault to true on the row, which points to the String CDT primitive and the xsd:string primitive map. Else if xsd:attribute/@type = ‘NormalizedStringType’, set isDefault to true on the row, which points to the String CDT primitive and xsd:string map. Else if xsd:attribute/@name = ‘listID’ or ‘listVersionID’ or ‘unitCodeListVersionID’, set isDefault to true on the row, which points to the NormalizedString CDT primitive and xsd:normalizedString map. Else if xsd:attribute/@type = ‘DateTimeType’, set isDefault to true on the row, which points to TimePoint CDT primitive and the xsd:token map. Else if xsd:attribute/@type = ‘IndicatorType’, set isDefault to true on the row that points to the Boolean CDT primitive and xsd:boolean map. Else if xsd:attribute/@type = ‘ValueType\_E7171E’, set isDefault to true on the row, which points to the NormalizedString CDT primitive and xsd:normalizedString map. Else if xsd:attribute/@name = ‘name’, set isDefault to true on the row, which points to the NormalizedString CDT primitive and xsd:normalizedString map. Else if contains(xsd:attribute/@type, ‘CodeContentType’) is true or xsd:attribute/@name = ‘listAgencyID’, set isDefault to true on all the row, which points to the Token CDT primitive and xsd:token map; and 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 false). |
| 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 false. |
| 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 false. |

#### Import BCCPs

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). Simple content means a complex type with simple content or simply a simple type.

Assuming the target xsd:element is a BCCP, the xsd:element/@type shall be a BDT that has already been imported. If it cannot be found the system should throw an error that indicates something is wrong either in the import logic or its implementation. To find an imported BDT, we can use a GUID of the type (or name, but it is easier to use the GUID).

##### 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. “Amount” for the DiscountAmount XSD element that uses the “Open\_ Amount” QBDT, “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 3.1.1.8 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 3.1.1.11.2 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 3.1.1.11.3) (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 3.1.1.11.4). If no match is found in either the ASCCP or BCCP table, the referred to entity is more likely an ASCCP although it can be a BCCP (because the assumption is that the BCCPs have already been imported in the earlier section from the Fields.xsd) and it has not been created. In this case, cascade to create the BCCP or the ASCCP first (follow the logic in 3.1.1.8 for BCCP in 3.1.1.11.1 for ASCCP) and then come back to create an ASCC record (follow the logic in 3.1.1.11.3).

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 3.1.1.11.1), then come back to create an ASCC (follow the logic in 3.1.1.11.3). Note: OAG uses mainly global elements. It uses only local-global (local element and global type) for the DataArea element. It 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 3.1.1.11.4).

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 3.1.1.11.5 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). Assign 0 if the context is an xsd:attribute.

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. If there is no BDT found report an error.

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.

#### Import other libraries

If another library of content is CCTS compliance and has convention similar to OAGIS, the easiest way to reuse the import code maybe is to re-organize the content of the library to have the same file and containment structure as OAGIS.

### 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 directly invoke:

BIEs Management: Create a top-level ABIE, Edit a top-level ABIE, Copy a top-level ABIE.

Business Context Management: Create a business context, Update a business context, and Discard a business context

Context Category Management: Create a context category, Update a context category, and Discard a context category

### Context Scheme Management: Create a new context classification scheme, Search and select a context classification scheme

A search UI should allow the user to look for an existing context scheme. A single search field should look through the Scheme\_ID, Name, Description, and Context\_Category\_ID (via the Name and Description columns of the Context\_Category table) columns of the Context\_Classification\_Scheme table.

A check box should be available so that the user can filter to only the latest version of the scheme.

The result dialog should show context category name, scheme name, scheme description, scheme version, and scheme agency name of each context scheme.

The UI should have an ‘OK’ button for the user to accept the selection; and, a ‘Cancel’ button which should return null. The ‘OK’ button should return a selected context scheme object or ID to the calling function.

Update a context classification scheme , Discard a context classification scheme

Code List Management: Create a new BIE code list, Create a new BIE code list from another (by extension & restriction, **Error! Reference source not found.**, and View/Edit a BIE code list.

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

Select a Business Context

Create a Business Context

Selected top-level ACC

Selected business context

Create a new business context

New business context

Create uncommitted BIE records/objects

Show top-level ABIE

Go to the Business Context Management and Show top-level ABIE sections for more detail.

##### Select a top-level concept

A list of top-level ASCCPs (note: the UI will just ask the user to select a concept) should be created for the user to select as a source for the top-level ABIE.

A top-level ASCCP is most likely a BOD in OAGIS. There should be a condition on the ASCCP records in the configuration file which indicates how to find the top-level ASCCPs. In the case of OAGIS, the condition is ‘Module != “Components” and Module != “Meta” and Module != “Noun” and Module != ‘Extensions’. The parameter name is TopLevelAsccpSql (see section 4.1). The reason to make this configurable is that this condition may change and a different condition may be needed for another OAGIS version, non-OAGIS library such as B2MML, or when a more relaxed condition is needed.

Make a query to select ASCCPs from the ASCCP table according to the TopLevelAsccpSql. This creates the top-level ASCCPs list. The user selects an ASCCP from the list. After that query for the ACC used by the ASCCP through the ASSCP.Role\_Of\_ACC column – this is the top-level concept.

##### Select a business context

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

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

##### Create uncommitted BIE records/objects

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 CreateTopLevelBIEs function.

###### Function: CreateTopLevelBIEs(tlASCCP, 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: tlASCCP = the given top-level ACC, tlACC = the given top-level ACC, bc = the given business context

Return: The top-level ABIE object.

CreateBIEs(tlACC, tlABIE)

tlASBIEP = CreateASBIEP(tlASCCP, tlABIE, null)

Return tlABIE

tlABIE = CreateABIE(tlACC, bc, true, null)

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

Description: Create uncommitted records of all descendant BIEs for the given ACC and its corresponding ABIE. This function is recursive.

Argument: gACC = the given ACC, gABIE = gACC’s corresponding ABIE, groupPosition = See explanation below.

The gABIE will be the owner (the association-from) of the child ASBIEs and BBIEs to be created.

The groupPosition is an integer data type. It is assigned a positive integer when the gACC is a “Semantic Group” (as indicated in the gACC.OAGIS\_Component\_Type column). If it is not a group, a -1 value is assigned. The positive integer is the sequencing key representing the position of the group within the gABIE. This is a peculiar case. Because the group is ignored in the BIE level, gABIE is not a corresponding BIE of the gACC but a parent of the gACC.

Note: gABIE.bc is the business context of the gABIE. The sk variable is the sequencing key for the ASBIE and BBIE. The sequencing key for ASBIE and BBIE will be of double data type because of the sequencing key of the children of the groups. Ex. If the group’s sequencing key is 3, then sequencing key of its children within the group’s parent are 3.1, 3.2, etc.

The skb (sequencing key base) variable takes care of the sequencing key particularly in the case when there are inheritances. Basically, the sequencing key of the extension has to be increased by the total number of children in the base. The represents the number of child associations (both BCC and ASCC) of an ACC, bACC[k].

This algorithm assumes there is not group within a group.

ASCC[ ] = QueryASCC(bACC[j])

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

Y

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

aASCCP = QueryASCCP( ASCC[i])

tACC = QueryTargetACC( aASCCP)

aASBIEP = CreateASBIEP(aASCCP, tABIE, gABIE)

CreateBIEs(tACC, tABIE, -1)

N

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

tACC is a Group?

CreateBIEs(tACC, **gABIE**, skb+ASCC[i].Sequencing\_Key)

Y

N

bACC[j-1] has a Base?

bACC[j] = queryBaseACC( gACC[j-1] )

Y

j = 1, bACC[0]=gACC

j = j+1

For each bACC[ j], **j=m…0**

Next bACC[j]

N

If groupPosition > 0 then

sk=groupPosition+ASCC[i].Sequencing\_Key\*0.1

Else

sk=(skb+ASCC[i].Sequencing\_Key).toDouble()

BCC[ ] = QueryBCC(bACC[j])

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

BBIE[i] = CreateBBIE(BCC[i], sk, gABIE, aBBIEP)

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

aBBIEP = CreateBBIEP(BCCP[i], gABIE)

If groupPosition > 0 then

sk=groupPosition+BCC[i].Sequencing\_Key\*0.1, Else

sk=(skb+BCC[i].Sequencing\_Key).toDouble()

Next BCC[i]?

Next ASCC[i]

Y

Y

N

N



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

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, pABIE = parent ABIE of this ABIE.

Return: An ABIE object

Populate the ABIE table as follows:

ABIE\_ID = Auto-generate database key.

ABIE\_GUID = Auto-generate GUID. Note see 3.11 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. If pABIE is not null, then use its Creation\_Timestamp.

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

State = “Editing”.

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, pABIE)

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, pABIE = the parent ABIE that causes the creation of this ASBIEP (may be null).

Return: An ASBIEP object

Populate the ASBIEP table as follows.

ASBIEP\_ID = Auto-generated database key.

ASBIEP\_GUID = Auto-generate GUID. Note see 3.11 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 = Current time if pABIE is not null; otherwise, pABIE.Creation\_Timestamp.

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

###### 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, seqKey, 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, seqKey = Sequencing key (position) of the BBIE, 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 3.11 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.

Sequencing\_Key = seqKey.

Cardinality\_Min = Inherit from the gASCC. The user can change this later.

Cardinality\_Max = Inherit from the gASCC. The user can change this later.

Nillable = 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 = pABIE.Creation\_Timestamp.

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

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

Description: Create uncommitted BBIE record for the given BCC.

Argument: gBCC = the given BCC for which BBIE should be created, seqKey = Sequencing key (position) of the BBIE, 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 3.11 Generate GUID about the GUID generation function.

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

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

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

BDT\_Primitive\_Restriction\_ID = Inherit through the chain gBCC -> BCCP -> DT -> BDT\_Primitive\_Restriction. BDT\_Primitive\_Restriction\_ID where the BDT\_Primitive\_Restriction.isDefault = true. The user can override later with another non-default BDT\_Primitive\_Restriction\_ID (i.e., assign another built-in type as represented by the BDT\_Primitive\_Restriction.CDT\_Primitive\_Expression\_Type\_Map\_ID column). Also if the next Code\_List\_ID column is not null, the user cannot override anything, i.e., it should not be editable because it means that the field is restricted to a code list in the next column and the user cannot relax that.

Code\_List\_ID = Inherit through the chain gBCC -> BCCP -> DT -> BDT\_Primitive\_Restriction.Code\_List\_ID if there is an associated row where the Code\_List\_ID is not null (otherwise this column will be null initially). The user can override later with a code list compatible with the default BDT\_Primitive\_Restriction\_ID (as identified in the previous column above) or a code list derived from the code list associated with this Code\_List\_ID.

Sequencing\_Key = seqKey

Cardinality\_Min = Inherit from the gBCC. The user can change this later.

Cardinality\_Max = Inherit from the gBCC. The user can change this later.

Default = null.

isNillable = 0.

Fixed\_Value = null.

isNull = 0.

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 = pABIE.Creation\_Timestamp.

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.

DT\_SC\_Primitive\_Restriction\_ID = Inherit through the chain gBCC -> BCCP -> DT -> DT\_SC -> BDT\_SC\_Primitive\_Restriction. BDT\_SC\_Primitive\_Restriction\_ID where the BDT\_SC\_Primitive\_Restriction.isDefault = true. The user can override later with another non-default BDT\_SC\_Primitive\_Restriction\_ID (i.e., assign another built-in type as represented by the BDT\_SC\_Primitive\_Restriction.CDT\_SC\_Primitive\_Expression\_Type\_Map\_ID column). Also if the next Code\_List\_ID column is not null, the user cannot override anything, i.e., it should not be editable because it means that the field is restricted to a code list in the next column and the user cannot relax that.

Code\_List\_ID = Inherit through the chain gBCC -> BCCP -> DT -> BDT\_SC\_Primitive\_Restriction.Code\_List\_ID if there is an associated row where the Code\_List\_ID is not null (otherwise this column will be null initially). The user can override later with a code list compatible with the BDT\_SC\_Primitive\_Restriction\_ID column above or a code list derived from the code list associated with this Code\_List\_ID.

Min\_Cardinality = Inherit from the associated DT\_SC record. The user can change this later.

Max\_Cardinality = Inherit from the associated DT\_SC record. The user can change this later.

Default = Null. The user can specify later.

Fixed\_Value = Null. The user can specify later.

Definition = Null. The user can specify later.

Remark = Null. The user can specify later.

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

###### Function: CreateBBIEP(gBCCP, pABIE)

Description: Create an uncommitted BBIEP record for the given BCCP

Argument: gBCCP = the given BCCP for the BBIEP should be created, pABIE = parent ABIE that causes the creation of this BBIEP (may be null).

Return: BBIEP object

Populate the BBIEP table as follows.

BBIEP\_ID = Auto-generated database key.

BBIEP\_GUID = Auto-generate GUID. Note see 3.11 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 if pABIE is not null; otherwise, it is pABIE.Creation\_Timestamp.

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

### Edit a top-level ABIE

This should invoke the Top-level ABIE search and select in section 3.3.7. Upon returning a top-level ABIE, go to the Show top-level ABIE section in 3.3.3. The state of the top-level ABIE selected for editing shall be returned to Editing.

### Show top-level ABIE

Create an expandable and collapsible table representation of the ABIE as shown in the Visualization section of the Requirement document. View Object of the top-level ABIE should be created. The user can:

1. Customize Business Term, Definition, and Remark of the top-level ABIE. Expand the top-level ABIE to shows its ASBIE and BBIE children.
2. Expand/Collapse a descendant ASBIE to customize its content.

The followings should be customizable: Cardinality min, Cardinality max, Nillable, ASBIE Definition and Remark, ASBIEP Definition and Remark, and (target) ABIE Business Term, Definition, and Remark.

The view object should derive these fields from its based CCs when the field values in the BIEs are Null, except the Remark, Nillable, and Business Term fields which do not exist in the CC.

1. Expand/Collapse a descendant BBIE to customize its content.

The followings should be customizable: The followings should be customizable: Cardinality min, Cardinality max, Nillable, Fixed Value (note: for the null fixed value, make the isNull true because Fixed Value = Null just means there is no fixed value), BBIE Definition and Remark, and BBIEP Business Term, Definition, and Remark.

The view object should derive these fields from its based CCs when the field values in the BIEs are Null, except the Nillable, Remark, Fixed Value, and Business Term fields which do not exist in the CC.

Regarding the BBIE.BDT\_Primitive\_Restriction\_ID and the BBIE.Code\_List\_ID columns. Corresponding to these two columns, the view object of the BBIE should have these fields as “Business Data Type” and “Primitive”. There may be multiple “Primitive” columns depending on the number of expression map columns in the CDT\_Allowed\_Primitive\_Expression\_Type\_Map table. The configuration file should be inspected for this information. See section 4.2 “Expression type mapping display configuration” for configuration detail. The user should be able to turn on/off non-default “Primitive” column. The default “Primitive” column (the built-in/native type of the expression) is indicated in the configuration file by the expressionTypeMapping@isDefault attribute. The “Primitive” column label should follow the element value in expressionTypeMapping/name element.

The “Business Data Type” column should show the name of the DEN of the BDT. This value should be derived from the BDT associated with the BBIE’s BBIEP’s BCCP.

When both the BBIE.BDT\_Primitive\_Restriction\_ID and BBIE.Code\_List\_ID columns are null, “Primitive” columns should show the name of the built-in type (e.g., XSD\_BuiltIn\_Type.Name column) or name of the code list (Code\_List.Name, this is because a BDT\_Primitive\_Restriction record can tie to a code list instead of a built-in type). It should be noted that the built-in type tables must follow the same column naming pattern. It will have a Name and a BuiltIn\_Type column, the latter is used for expression generation (so a static SQL should work, it only needs to look up the primary key column name). When both the BBIE.BDT\_Primitive\_Restriction\_ID and BBIE.Code\_List\_ID columns are null, the built-in type/code list to show is from the joining between the BDT\_Primitive\_Restriction table and the CDT\_Allowed\_Primitive\_Expression\_Type\_Map table that match the BDT\_ID of the BBIE->BBIEP->BCCP and the BDT\_Primitive\_Restriction.isDefault = true. It should be noted that the user can override this default as follows.

The BBIE.BDT\_Primitive\_Restriction\_ID column allows the user to select from a list of BDT\_Primitive\_Restriction. BDT\_Primitive\_Restriction\_ID. The *allowable list* should be obtained from querying the BDT\_Primitive\_Restriction table for rows that match the BDT of the BBIE->BBIEP->BCCP similar to in the previous paragraph but disregard the BDT\_Primitive\_Restriction.isDefault column. If the user wants to use a code list that is not in the allowable list, BBIE.BDT\_Primitive\_Restriction\_ID column should be left null and the BBIE.Code\_List\_ID column will be used. Any code list which is based on the code list in the allowable list can be used. If the default on the BDT\_Primitive\_Restriction is just a code type without a specific code list, then any code list can be used. The usage of the BBIE.BDT\_Primitive\_Restriction\_ID or BBIE.Code\_List\_ID column may be made transparent to the user, i.e., a combo-box can contain a choice that takes the user to select a code list not in the allowable list when it is logically permissible.

Expanding the BBIE shows its supplementary component (SC) from the BBIE\_SC table, if any.

The followings should be customizable for SC: Cardinality max (this can be represented as a toggle button b/c it is either on or off), Fixed Value, Default, Definition, Business Term, and Remark.

The view object should derive these fields from its based DT SC when the field values in the BBIE are Null, except the Fixed Value, Default, Business Term, and Remark fields which do not exist in the CC.

The behavior associated with the BBIE\_SC.DT\_SC\_Primitive\_Restriction\_ID and the BBIE\_SC.Code\_List\_ID columns is similar to the behavior associated with the BBIE.BDT\_Primitive\_Restriction\_ID and BBIE.Code\_List\_ID columns discussed above. It is only that in this case the relevant tables are BDT\_SC\_Primitive\_Restriction and CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map. These tables use the same built-in types table(s). The configuration detail in section 4.2 also applies to this case. Since SCs do not use DT, show the DT\_SC.Representation\_Term instead of the “Data Type” column in the view.

The following rules apply to all customizations above.

The customized Cardinality min can only be more than the original value. The customized Cardinality max can only be less than the original value. The user should be able to reset these customized values to the original/derived ones.

On this Edit top-level ABIE page, there should be buttons that allow the user to save, discard, publish the top-level ABIE. They respectively invoke the logic in the Save a top-level ABIE in section 3.3.4, Discard a top-level ABIE in section 3.3.5, and Publish a top-level ABIE in section 3.3.6. Use optimistic concurrency control strategy for all these transactions.

A ‘Close’ button should also be available to bring the user back to the Application Home Screen. A warning should be given that unsaved change will be discarded and ask the user for a confirmation.

### Save a top-level ABIE

Ensure that the State of the top-level ABIE is Editing, and serialize and commit the top-level ABIE and related descendant BIEs to the database. If the top-level ABIE is not in the Edit state, show an error message.

### Discard a top-level ABIE

Warn the user that the top-level ABIE and descendant BIEs will be permanently removed from the database. Upon confirmation, remove the top-level ABIE and related descendant BIE records from the database. Return to the Application Home Screen upon successful execution.

### Publish a top-level ABIE

Serialize and commit the top-level ABIE and related descendant BIEs to the database. State of top-level ABIE should be written as Published. Return to the Application Home Screen upon successful commit.

### Top-level ABIE search and select

This should return a top-level ABIE (or ABIE\_ID). Return null if the user cancel.

This page should allow the user to search for the top-level ABIE. The search field should look through the following columns: DEN of the associated ACC, ABIE’s Definition, ACC’s Definition, and ABIE’s Business\_Term.

The search result should show the following details of the top-level ABIE: ACC’s DEN, ABIE.Business\_Term, applicable Business Contexts, ABIE.Definition, ABIE.Last\_Update\_Timestamp, ABIE’s last updated by user, ABIE’s client, ABIE.Version, ABIE.Remark, ABIE.Status.

### Copy a top-level ABIE

This should invoke the Top-level ABIE search and select in section 3.3.7. Then invoke the Search and Select a Business Context in section 3.4.1. Clone the selected top-level ABIE and its entire descendant BIEs. Associate the new top-level ABIE with the selected BC. Then invoke the Show top-level ABIE section in 3.3.3 with the new top-level ABIE.

Note that the cloned BIEs should have different GUIDs. The WHO columns should be updated appropriately. The State column of the top-level ABIE should be set to Editing.

### Extend a top-level ABIE

This use case will happen as part the process of editing a top-level ABIE. The user should be able to invoke the extension from the BIE editing window.

After clicking (or right clicking to bring up a context menu see Figure 1) on an Extension element, while the top-level ABIE is in the Editing state, the user should be able to access the Create ABIE Extension Locally or the Create ABIE Extension Globally function. Upon entering the Create ABIE Locally function, the following logic flow below should be followed. If the user selects the Create ABIE Extension Globally, the logic is the same - it is only that the entity names are different. The differences will be noted in the table detail in each subsection.

Profile BOD (Top-level ABIE Page

AckFields

DataArea

AA

Sender

Extension

AnyExtension

Amount

The user right click here to show the context menu.

Note: AA standards for ApplicationArea

Figure : Starting point on the BIE page for the user extension

In the following subsections, let the variable tlABIE refer the top-level ABIE in context and the variable eABIE refer to the extension ABIE in context, e.g., ApplicationAreaExtensionType, PurchaseOrderExtensionType, AllExtensionType.

Let’s also designate the CC corresponding to the tlABIE as tlACC.

Let the eACC be the ACC corresponding to the eABIE. In order that the OAGIS model content never be put in the Editing state additional associations the user adds to an eACC are stored in a separate ACC. This separate ACC is of component type User Extension Group. It behaves like an XSD Group. The figure below explains this …… Let the User Extension Group ACC corresponding to the eACC be ueACC.

**(1)** Get the latest revision of the ueAcc.

**(2)**

Check state and ownership of ueAcc

ueAcc = null?

Go to Editing User Extension Group ACC use case.

Indicate to the user that the eACC is being extended by another user. Indicate which user. The user can acknowledge and then return to top-level BIE editing page.

Indicate to the user that the eACC is in Candidate state and ask whether the user wants to switch the state back to Editing.

Change ueACC state and its association to Editing.

Return to the previous top-level ABIE editing page

Indicate to the user that the eACC is being extended by another user. Indicate which user. It is available for review. Ask the user whether he’d like to review the extension.

Go to Review User Extension Group ACC use case.

Indicate to the user that the eACC was last extended by who. Ask the user to whether he would like to create a new revision.

**#a**

**#b**

Y

N

N

**#c**

**#d**

Y

**(3)** Go to Create a New User Extension Group ACC use case

N

Y

N

Y

**(4)** Go to Create a New User Extension Group ACC Revision use case

**#e**

Figure : Logic flow for creating a local ABIE extension

1. This is done by querying for child ASCC of the eAcc and then get the target ACC via the relationship ASCC.Assoc\_To\_ASCCP\_ID.Role\_Of\_ACC\_ID. From the obtained target ACC, gets its latest revision from the ACC table. The target ACC must have the ACC.OAGIS\_Component\_Type = ‘4’ (User Extension) with the highest version number as indicated by the ACC.revision\_num column.
2. If ueAcc state is in the Editing state and the current user is the owner, go to branch **#a**. If ueAcc is in Editing state but the current user is not the owner, go to branch **#b**. If the ueAcc state is Candidate and the current user is the owner, go to branch **#c**. If the ueAcc state is Candidate and the current user is not the owner, go to branch **#d**. If the ueAcc is in the Published state, go to branch **#e**.
3. Create a New User Extension Group ACC
4. Create a New User Extension Group ACC Revision

#### Create a New User Extension Group ACC

A user extension group ACC is an ACC whose OAGIS\_Component\_Type = 4. An uncommitted ACC, ASCCP, and ASCC current and history records in a transaction should be created according to the subsections below.

A screen should pop up to ask the user for the Module and Namespace to which the extension should belong.

Then, a UI that allows the user to edit the extension should be displayed so that the user can create the extension content, see section 3.3.9.3 Edit User Extension Group ACC for UI requirement. If the user cancel this creation, the whole transaction should be rolled back.

##### Populate an ACC current record

This is the new ueACC being created.

|  |
| --- |
| ACC\_ID = Auto-generate database key. |
| ACC\_GUID = Create a new GUID. |
| Object\_Class\_Term = Follow this pattern. If the Object\_Class\_Term of the eACC is “Application Area Extension” the Object\_Class\_Term for this ACC is “Application Area User Extension Group”. I.e., insert the “User” before the “Extension” and suffix the end with “Group”. So for “All Extension”, it would be “All User Extension Group” and for “Attached Item Extension”, it would be “Attached Item User Extension Group”. |
| DEN = Object\_Class\_Term + “. Details”. |
| Definition = “A system created component containing user extension to the ” + eACC.Object\_ClassTerm. |
| Based\_ACC\_ID = Null. |
| Object\_Class\_Qualifier = Always empty. |
| OAGIS\_Component\_Type = 4 for “User Extension Group”. |
| Module = Get from the user input earlier. |
| Namespace\_ID = Get from the user input earlier. |
| Created\_By\_User\_ID = The logged in user. |
| Owner\_User\_D = The logged in user. |
| Last\_Updated\_By\_User\_ID = The logged in user. |
| Creation\_Timestamp = Current time. |
| Last\_Update\_Timestamp = Same as Creation\_Timestamp. |
| State = 1. **Note:** 1 means Editing. |
| revision\_num = 0. |
| revision\_trackiing\_num = 0. |
| Revision\_Action = Null. |
| release\_id = Null. |
| Current\_ACC\_ID = Null. |

##### Populate an ASCCP current record

|  |
| --- |
| ASCCP\_ID = Auto-generate database key. |
| ASCCP\_GUID = Create a new GUID. |
| Property\_Term = Same as the Object\_Class\_Term of the ACC above. |
| Definition = “A system created component containing user extension to the ” + eACC.Object\_ClassTerm. |
| Role\_Of\_ACC\_ID = Foreign key to the ACC above. |
| DEN = Property\_Term + “. ” + First(ACC(Role\_Of\_ACC\_ID).DEN). |
| Created\_By\_User\_ID = The logged in user. |
| Last\_Updated\_By\_User\_ID = The logged in user. |
| Creation\_Timestamp = Current time. |
| Last\_Update\_Timestamp = Same as Creation\_Timestamp. |
| State = 4. **Note:** 4 means published. |
| Module = Get from the user input earlier. |
| Namespace\_ID = Get from the user input earlier. |
| Reusable\_Indicator = False. |
| revision\_num = 0. |
| revision\_trackiing\_num = 0. |
| Revision\_Action = Null. |
| release\_id = Null. |
| Current\_ASCCP\_ID = Null. |

##### Populate an ASCC current record

|  |
| --- |
| ASCC\_ID = Auto-generate database key. |
| ASCC\_GUID = Generate a new one. |
| Cardinality\_Min = 1. |
| Cardinality\_Max = 1. |
| Sequencing\_Key = 1. |
| Assoc\_From\_ACC\_ID = eACC. ACC\_ID. |
| Assoc\_To\_ASCCP\_ID = ASCCP\_ID of the above ASCCP record. |
| DEN = eACC.Object\_Class\_Term + “. “ + ASCCP(Assoc\_To\_ASCCP\_ID).DEN |
| Definition = “System created association to the system created user extension group component.” |
| Created\_By\_User\_ID = The logged in user. |
| Owner\_User\_ID = The logged in user. |
| Last\_Updated\_By\_User\_ID = The logged in user. |
| Creation\_Timestamp = Current time. |
| Last\_Update\_Timestamp = Same as Creation\_Timestamp. |
| State = 4. **Note:** 4 means published. |
| revision\_num = 0. |
| revision\_trackiing\_num = 0. |
| Revision\_Action = Null. |
| release\_id = Null. |
| Current\_ASCC\_ID = Null. |

##### Populate an ACC history Record

The data should be the same as that in the ACC current record except below columns.

|  |
| --- |
| ACC\_ID = Auto-generate database key. |
| Based\_ACC\_ID = ACC\_ID of the ACC current record. |
| revision\_num = 1. |
| revision\_trackiing\_num = 1. |
| Revision\_Action = 1. |
| release\_id = Null. |
| Current\_ACC\_ID = ACC\_ID of the ACC current record. |

##### Populate an ASCCP history record

The data should be the same as that in the ASCCP current record except below columns.

|  |
| --- |
| ASCCP\_ID = Auto-generate database key. |
| revision\_num = 1. |
| revision\_trackiing\_num = 1. |
| Revision\_Action = 1. |
| release\_id = Null. |
| Current\_ASCCP\_ID = ASCCP\_ID of the ASCCP current record. |

##### Populate an ASCC history record

The data should be the same as that in the ASCC current record except below columns.

|  |
| --- |
| ASCC\_ID = Auto-generate database key. |
| revision\_num = 1. |
| revision\_trackiing\_num = 1. |
| Revision\_Action = 1. |
| release\_id = Null. |
| Current\_ASCC\_ID = ASCC\_ID of the ASCC current record. |

#### Create a New User Extension Group ACC Revision

In this case, the State of the ueACC current record and all its associations (BCC and ASCC) must be updated to Editing. Then, the corresponding history records must be created with an incremental revision\_num from the latest revision. The revision\_trackiing\_num is starting at 1. The Revision\_Action is 2 (which is update).

Then, a UI that allows the user to edit the extension should be displayed, see section 3.3.9.3 Edit User Extension Group ACC for UI requirement. If the user cancels this creation, the whole transaction should be rolled back.

#### Edit User Extension Group ACC

This section outline UI requirements for editing a user extension.

### The UI should show a tree model of the Model BOD (CC) correspondnig to the tlABIE. All nodes except the Extension element corresponding to the eACC should be uneditable. Uneditable nodes may be rendered in a different background. In Figure 3, for example, uneditable nodes are rendered with the blue background. Note that since this is a CC rendering the ACC whose component type is 3 (Semantic Group) will be rendered; so as the type extension. See 3.10.1 for basic Architecture for managing revisions

#### Current and history records

Logically, CCs revisions and releases are managed in two sets of records – current and history. This can be viewed as two sets of tables as shown in Figure 12 below. However, both current and history records are implemented with only a single set of tables; and the current and history records are differentiated by the revision\_num column. The current record always have a revision\_num of 0, while the history record has a revision\_num of 1 or higher. The history record always points to its corresponding current (master) record.



Figure 12: Concepts of current and history records

Everytime there is a change to a CC, a history record is inserted. For a new revision, the revision\_num is incremented from the latest revision. For every change during the revision process, the history records are inserted with the same revision\_num but differing revision\_tracking\_num. The revision\_tracking\_num supports a machinsm for the user to undo his/her changes. CCs have a state which can be Editing, Candidate, Published, and Deprecated. Once a revision of a CC is published, the history record whose revision\_tracking\_num is 1 or more is deleted except the highest (latest) one to reflect the final picture of the CC’s revision.

The history record also has a revision\_action column, which indicates the database transaction that has happened to the current record. The value can semantically means insert, update, and delete. The purpose of this column is two-fold. First, it supports undoing changes during the revision. The other is to support the release of the CC content. This is particularly important when the action is ‘delete’. When an association is deleted, it is no longer present in the current record. Therefore, the history record indicates that the association is no longer exist in releases after the deletion has occurred (this works in conjunction with the release\_id column).

CC tables have a common set of columns, such as the revision\_num and others discussed above, to keep track of changes, revision, and release. See the data model for the definition of these columns and how they should be populated in general. Unless otherwise specifically specified in other subsections of section 3.10, explanations in the data model and in this 3.10.1 section shall be the guideline for populating these revision related columns.

#### CC states

All CC entities have a State column that keeps track of its life-cycle. The state can be Editing, Candidate, Published, and Deprecated. An ASCC and BCC can also have a Deleted state, but it is not represented by the State column.

Editing state means that the entity is being edited. At a particular point in time, an entity can be edited by only one user designated by the entity’s owner\_user\_id column. The current owner can assign another another user as owner to replace him/herself. The Editing state facilitates a work-in-progress situation. That is, while an entity is in the Editing state only the owner can see the entity.

The owner can move the entity from the Editing to the Candidate state. The Candidate state facilitates a draft review. This is when the entity can be visible and reviewed by other users. Comments may be given by other users. The owner is not allowed to make changes to an entity in the Candidate state; however, he/she can move the entity back to the Editing state to make changes.

From the Candidate state, the owner can move the entity to the Published state. Entities in Published states are stable and reusable by other entities. Once the owner move an entity into the Published state, he has relingquished his control and another user can create a new revision of that entity.

An entity may be deprecated after published, then it would have a Deprecated state. Deprecated entities must be restricted from reuse. In other words, no new references to the deprecated entity shall be made. The State 4 column in Table 2 shows an example that the Address ACC happens to be deprecated during the editing of the Broker Party ACC. In such situation, the owner of the Broker Party ACC shall be notified.

Dependencies between entities means that there is a restriction on the states of dependent entities. Generally, the referring entity shall not have a state that is more advanced than the referred to entity. For example, assuming a user is building a new Broker Party ACC. It has an ASCC that uses a new ASCCP Office Address which in turn reuses the Address ACC. During the process of building the Broker Party ACC, Table 2 below displays states of these entities at different points in time. The columns, State 3 and State 5, illustrate invalid situation. In State 3, the Broker Party ASCCP has a Candidate state, which is more advanced than the Editing state of the Broker Party ACC it is referring to. In such situation, when reviewing the Broker Party ASCCP, users cannot see the content of the Broker Party ACC. In State 5, the Broker Party ASCCP is in a Published state, which is again more advanced than the Candidate state of the Broker Party ACC. In this case, the Broker Party ASCCP cannot really in a reusable state, because its actual data structure, defined in the Broker Party ACC, may change. State 5 column has another problem in that the state associated with the Broker Party. Office. Address ASCC has more advanced state than that of the Office Address ASCCP. In this case, users cannot review the Broker Party ACC that is in the Candidate state because the content it is dependent on, the Office Address ASCCP, is not accessible.

Since an entity can shuttle back and forth between Editing and Candidate state, this presents another requirement. When switching an entity state from Candidate back to Editing, the system has to check if there is a referring entity that would be having a more advanced state after the switch back.

An exception to the above rule has to do with the Deprecated state. The referring entity can be deprecated before the referred to entity, given that both entities are in the Published state before the referring entity is deprecated. For example, an ASCCP can be deprecated while the ACC it refers to is not. Similarly, an assocition can be deprecated while the ASCCP or BCCP it refers to is not.

Table 2: Example CC states dependency

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Entity | State 1 | State 2 | State 3 | State 4 | State 5 | State 6 |
| Broker Party ASCCP | Editing | Editing | *Candidate* | Editing | *Published* | Editing |
| Broker Party ACC | Editing | Editing | Editing | Candidate | Candidate | Published |
| Broker Party. Office. Address ASCC | Editing | Editing | Editing | Candidate | *Candidate* | Published |
| Office Address ASCCP | Editing | Published | Editing | Published | Editing | Published |
| Address ACC | Published | Published | Published | Deprecated | Published | Published |
| Address. Address Line. Address Line BCC | Published | Published | Published | Published | Published | Published |
| Address Line. Seq Text BCCP | Published | Published | Published | Published | Published | Published |
| Seq Text. Type BDT | Published | Published | Published | Published | Published | Published |
| Address. Geo Cordinate. Coordinate ASCC | Published | Published | Published | Deprecated | Published | Published |
| Geo Coordinate. Coordinate ASCCP | Published | Published | Published | Deprecated | Published | Published |
| Coordinate ACC | Published | Editing | Candidate | Deprecated | Published | Published |

An ASCC and BCC can additionally have a Deleted state. No other kinds of entities cannot have a Deleted state (once created, they cannot be deleted). Deleted state is not represented by the State column but a Revision\_Action column. Other kinds of entities are implicitly deleted when they are not used in any association. An ASCC or BCC can only be transitioned into the Deleted state if it is in the Published or Deprecated state.

#### Summary of state related business rules

|  |  |
| --- | --- |
|  | Referring entity cannot have a more advance state than the referred to entity, with one exception. The referring entity can be deprecated before the referred to entity, given that both entities are in the Published state before the referring entity is deprecated. For example, an ASCCP can be deprecated while the ACC it refers to is not. Similarly, an assocition can be deprecated while the ASCCP or BCCP it refers to is not. |
|  | Only the owner can make changes to the entity. |
|  | There can be only one active revision of an entity at a particular point in time. I.e., there cannot be two revisions of an entity in Editing or Candidate state. I.e., A new revision can only be created on an entity in the Published state or Deprecated state. |
|  | A new revision cannot be created on a deprecated entity. Does this make sense??? |
|  | No new reference to a deprecated entity shall be allowed. I.e., a deprecated entity shall no longer be reused. |
|  |  |

#### Unit of control and concurrency management

Because CCs are typically interconnected, this section describes the method to manage the concurrency of accessing entities. The explanation about CC states in the previous section provides pre-requisite for concurrency control.

For concurrency control, units of control (or unit of check out) must be first defined. They are as follows:

1. ACC Unit of Control (UoC): States of an ACC and its child ASCC and BCC are always checked out altogether. This means that their states shall be transitioned together.
2. ASCCP UoC: An ASCCP can be checked out independently.
3. BCCP UoC: An BCCP can be checked out independently.
4. BDT UoC: A BDT can be checked out independently.

Owners of all active UoCs (UoCs in the Editing state) must be notified of a state change event associated with these UoCs. A mechanism may be created to filter events such that only the owners of UoCs that are directly and indirectly referring and referred to by the UoC generating the event are notified.



Figure 13: Example core components

Taking the core components in Figure 13 as an example, let’s assume that the PartyScreenResponseType ACC is first checked out into the Editing state by a user – U1. Then, another user, U2, checked out the CoordidateType ACC into the Editing state. U1 shall be notified of that event.

Next, U1 transitioned the PartyScreenResponseType ACC into the Candidate state. U2 shall be notified of that event. U2 should review the PartyScreenResponseType ACC to ensure that the changes do not overlap with the changes he was going to make to the CoordinateType ACC.

Next, U2 moved the CoordinateType ACC into the Candidate state. Similarly, U1 shall be notified of the event and review the content in the CoordinateType ACC to ensure that there would be no overlap with the changes he is going to make to the PartyScreenRespsonseType ACC.

#### Release management

This is a future version feature.

### CCs and DTs management home screen

The CCs and DTs management home screen is where all the CCs and DTs management functionalities can invoked. This includes 3.10.4 View CCs, 3.10.5 Create a new ACC (and an ASCCP), 3.10.6 Create a new revision of an ACC, 3.10.7 Edit an ACC, 3.10.13 Create a new ASCCP, 3.10.14 Create a new revision of an ASCCP, 3.10.15 Create a new BCCP, and 3.10.16 Create a new revision of a BCCP.

ACC and ASCCP content rendering. In this screen, the user can expand any components to inspect their content.

In the figure, the association to the Broker ASCCP is added to the extension. The dotted association line between the Extension and the Broker nodes differentiate the user extension content from other OAG built-in extension content. In this example, the Broker node is shown as uneditable because it is a reuse of an OAGIS standard CC. On the other hand, the IoTID is shown as editable because it is a CC created by a user that is not an OAGIS Developer (i.e., Created\_By\_User\_ID column of the IoTID points to the user whose the User.OAGIS\_Developer = False or 0).

The figure also shows the type inheritance hierarchy for the AA Extension ACC. It also illustrates that there is no extension content in the All Extension ACC.

Menus must exists for the following functionalities: Add an association, Create a new ACC, Create a new ASCCP, Create a new BCCP. See more design detail in section 3.10 for these actions. These functionalities should be provided to the user on a pop-up window so that the user does not have to leave the current user extension screen. One way to provide the user access to these functionalities is by right clicking on the All Extension ACC or the AA Extension ACC box. It should be noted that depending on the state and ownership of the All Extension ACC User Extension Group and the AA Extension ACC User Extension Group, the user may or may not have access to these and the below functionalities.

The user must also have access to the following functionalities.

* Delete an existing association. E.g., the user must be able to delete an association to the IoTID BCCP. Note that this does not delete the IoTID BCCP but the BCC association to BCCP. This may be offered to the user as a context menu when right clicking on a particular component. See 3.10….
* Make edit to details of existing associations. See 3.10…
* For a component that is a user defined component (such as the IoTID BCCP in the example below), the user should be able to make a new revision or modification depending on the state and ownership of the component. In addition, if the component is a user defined ASCCP, which may contain other user defined components. The user should be able to revise or modify those descendants user defined components as well. Access to these functionalities may be offered through double-clicking or right clicking on the component that shows possible actions. See section 3.10 for details relevant to these possible actions.

Open User Area ACC

All Extension ACC

AA Extension ACC

AckFields

DataArea

AA

Sender

Extension

AnyExtension

Amount

Broker

IoTID

Figure : A tree model in the UI for user extension editing

#### Review User Extension Group ACC

Append a child BBIE or the Append a child ASBIE functionalities below.

The user should also be able to access the Append a child BBIE and the Append a child ASBIE functionalities when clicking on one of the ASBIE user extensions (any descendant ASBIE that is derived from the user ACC[[1]](#footnote-1), basically the chain of user ASCC->user ASCCP->user ACC). In addition, the user should be able to access the Insert a sibling BBIE and the Insert a sibling ASBIE functionalities below when clicking on one of the ASBIE or BBIE user extensions.

In the following subsections, the term “context ASBIE” is used to referred to the ASBIE selected by the user, i.e., the ASBIE parent to which the child ASBIE, child BBIE, sibling BBIE, or sibling ASBIE will be added. Since an ASBIE should have associated ASBIEP and ABIE and also corresponding ASCC, ASCCP, and ACC from which it is derived, we use the terms “context ASBIEP”, “context ABIE”, “context ASCC”, and so on, to refer to these entities. And since in some cases a CC entity is a user CC, the terms “context user ASCC”, “context user ASCCP, and so on are also used.

#### Append a child BBIE

The user can add a BBIE by reusing an existing BCCP or creating a new one.

A UI allowing for searching an existing BCCP to be reused should be shown. The search should look up BCCP.Property\_Term, BCCP.Representation\_Term, BCCP.DEN and BCCP.Definition. On that UI, the option for creating a new BCCP should be available. From the UI, the user should be able to cancel and go back to the main top-level ABIE editing page. More detailed logic about the two options is discussed in subsections below.

##### Reuse a BCCP

The search page should only show BCCPs that are in reusable states, which include the ‘Published’ and ‘Deprecated’ states.

If the user found an existing BCCP to be reused:

1. A user BCC record must be created according to the detail below.
2. Create a BBIEP record according to the detail below.
3. Create a BBIE record according to the detail below.

Once all necessary records are created, the UI shows the BBIE in the tree and its properties on another pane that the user can customize just like the typical BBIE customization functionality.

###### Detail of the BCC record creation

BCC\_ID = Auto-generated database key.

BCC\_GUID = Auto-generated database key.

Cardinality\_Min = 0.

Carinality\_Max = 1.

Assoc\_To\_BCCP\_ID = the BCCP\_ID the user has selected for reuse.

Assoc\_From\_ACC\_ID = The ACC\_ID of the context ACC.

Sequencing\_Key = The number of ASCC and BCC children of the context ACC plus one.

Entity\_Type = 1.

DEN = First(Context ACC DEN) + “. ” + DEN(Selected BCCP)

###### Detail of the BBIEP record creation

Follow the CreateBBIEP function in 3.3.1.1.3.11 where gBCCP is the BCCP the user selected for reuse and pABIE is the context ABIE. The followings are exceptions.

* Creation\_Timestamp = current time.

###### Detail of the BBIE record creation

Follow the CreateBBIE function in 3.3.1.1.3.10 where gBCC is the BCC created above, pABIE is the context ABIE, tBBIEP is the BBIEP created above, the seqKey is the next integer larger than the last child of the pABIE. The followings are exceptions.

* Creation\_Timestamp = current time.

##### Create a new BCCP

In this case, UI showing BCCP data fields to be populated by the user should be invoked. The user should be able to populate the following data field below. Once finished, a BCCP record should be created in the database along with the BBIEP and BBIE records as described in 3.3.11.2.1 above.

The user should populate the following BCCP data fields corresponding to the columns in the BCCP table - Property\_Term, Definition, and BDT\_ID. Representation\_Term and DEN are also shown in the UI but are not editable. The former is derived from the selected BDT. DEN is then derived from the Property\_Term and the Representation\_Term. To assign the BDT\_ID, a UI should allow the user to search for the BDT (using both the DT.Data\_Type\_Term column and DT.Definition column, also note that the selection condition must be DT.DT\_Type = 1). The UI should also allow a filter by version number. The user can only reuse an existing BDT. However, further restriction can be made on the BBIE UI. The user should be able to inspect the detail of the BDT when selecting a BDT. The UI for displaying the detail of the BDT is discussed in the next section.

The timestamp columns of BCCP should reflect the current time.

The Created\_By\_User\_ID column should reflect the current user.

The State column should be in the Published state right away.

##### Displaying BDT

The BDT UI should show the DT.DEN, DT.Definition (initially collapsed), DT.Revision\_Documentation (initially collapsed), Default BDT\_Primitive\_Restriction -> CDT\_Primitive\_Expression\_Type\_Map -> Code\_List or [Default Built-in Type per the configuration described in 4.2]. If it is a code list, shows the Name, List\_ID, List\_Agency\_ID, and Version\_ID of the code list.

SCs of the BDT should also be shown. For each SC, show the DT\_SC.Property\_Term, DT\_SC.Representation\_Term, DT\_SC.Min\_Cardinality, DT\_SC.Max\_Cardinality, DT\_SC.Definition (initially collapsed).

#### Append a child ASBIE

The user can use an existing ASCCP or create a new one.

A UI allowing for searching an existing ASCCP to be reused should be shown. The search should look up ASCCP.Property\_Term, ASCCP.Representation\_Term, ASCCP.DEN, ASCCP.Definition, ACC.Object\_Class\_Term, ACC.DEN, and ACC.Definition. On that UI, the option for creating a new ASCCP should be available. From the UI the user should be able to cancel and go back to the main top-level ABIE editing page. More detailed logic about the two options is discussed in subsections below.

##### Reuse an ASCCP

The user can only reuse an ASCCP created by the user in the same group and those created by a user in the ‘CC Developer’ group. The actual name of the CC Developer group is a configuration parameter, see **Error! Reference source not found.**. In addition, the ASCCP must be in the ‘Published’ state.

#### Delete a child BBIE or ASBIE

#### Move a child BBIE or ASBIE (currently out of scope)

This function allows moving the BBIE or ASBIE child of the extension. Sequencing keys among children have to be adjusted accordingly.

## 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 and select a business context

This should return a business context (or Business\_Context\_ID) or return null if the user cancel.

#### Bottom search

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

#### Top search

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

The user can click a ‘Cancel’ or an ‘OK’ button. The ‘Cancel’ button returns null. The ‘OK’ button returns the selected business context object or Business\_Context.Business\_Context\_ID.

### Create a business context

This should return a business context (or Business\_Context\_ID) or return null if the user cancel.

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 4. When the user clicks the ‘Add’ button in Figure 4, it brings up Figure 5 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 5 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.

The ‘Close’ button brings the user back to the Application Home Screen. A warning should be given that unsaved changes will be discarded and ask the user for a confirmation.



Figure : Search and Select Context Value screen

### Show a business context

A UI like shown in Figure 6 below should be provided. The ‘Delete’ button can delete each row of the business context value. The ‘Add’ button invokes the UI in Figure 5. The ‘Save’ button, update related records in the Business\_Context and Business\_Context\_Value tables and the concurrency control rule is in effect here. The ‘Close’ button takes the user back to the Application Home Screen. The ‘Close’ button should ask for confirmation from the user first, saying that “Unsaved change will be discarded.”



Figure : Show a Business Context screen

### Update a business context

Invoke the use case - Search and select a business context. If a business context object that is not null is return, invoke the use case - Show a business context; otherwise, return to the Application Home Screen.

### Discard a business context

Invoke the use case - Search and select a business context. If a business context object returned is not null, show a confirmation screen warning that the business context will be permanently removed; otherwise, return to the Application Home Screen. If the user cancels in the confirmation screen, invoke again the use case - Search and select a business context. If the user confirms, check if the business context is used by any ABIE. If the business context is used by an ABIE, show the message “The selected business context cannot be discarded. The %ABIEs% with the following IDs depend on it. They need to be discarded first.” List the ABIE.ABIE\_ID after the message. If the business context is not used by any ABIE, remove the related records in the Business\_Context and Business\_Context\_Value tables from the database. Upon completion, return to the Application Home Screen.

Note: The %ABIEs%, refers to a menu configuration variable in the Configuration File.

## Context Category Management

### Create a context category

The user creates a Context Category by specifying the Name and Description corresponding to the columns in the Context\_Category table. The UI allows save and cancel actions. The save action writes information into the Context\_Category table and takes the user back to the Application Home Screen. The cancel action takes the user back to the Application Home Screen.

### Search and select a context category

A UI should allow the user to search the context category by Context\_Category.Name and Context\_Category.Description. An ‘OK’ button returns a selected context category or null, if none is selected, is returned. A ‘Cancel’ button takes the user back to the Application Home Screen.

### Update a context category

Invoke the use case - Search and select a context category. If null is returned, go back to the Application Home Screen. If a context category is returned, invoke the use case - Show a context category.

### Show a context category

A UI such as shown in Figure 7 below should allow the user the edit the context category. The ‘Save’ button, update the record in the Context\_Category table and the concurrency control rule is in effect here. The ‘Close’ button takes the user back to the Application Home Screen. The ‘Close’ button should ask for confirmation from the user.



Figure : Show a context category screen

### Discard a context category

Invoke the use case - Search and select a context category. If a context category object that is not null is returned, show a delete confirmation screen warning that the context category will be permanently removed; otherwise, return to the Application Home Screen. If the user cancels the delete confirmation, invoke again the use case - Search and select a context category. If the user confirms the delete, check if the context category is used by any context scheme. If the context is used by a context scheme, show the message “The selected context category cannot be deleted. The context schemes with the following IDs depend on it. They need to be deleted first.” List the Context\_Scheme.Context\_Scheme\_ID after the message. If the business context is not used by any context scheme, delete the selected record in the Context\_Category table from the database. Upon completion, return to the Application Home Screen.

## Context Classification Scheme Management

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

### Create a new context classification scheme

The user specifies values corresponding to columns in the Context\_Classification\_Scheme table. The column Name, Agency\_ID, Version\_ID, 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 such as shown in Figure 8 should be present to the user. The ‘Close’ button takes the user back to the Application Home Screen. A warning should be given that unsaved changes will be discarded and ask for a confirmation from the user. A context classification scheme object should be returned if a new context classification scheme has been created; otherwise, return null.

The ‘Save’ button creates new records in the Context\_Classification\_Scheme and Context\_Classification\_Scheme\_Value tables. Before creating the record the following validation should be performed in addition to all required fields being specified.

* Potential for unintentionally creating a classification scheme with the same identity but different names: Is there a Context\_Classification\_Scheme record with the same Scheme\_ID and Scheme\_Agency\_ID?
  + If so, check if Version\_ID is the same as any of those existing records.
    - If so, give an error message that the same version cannot be created.
    - If not, check if Scheme\_Name is the same as the existing records.
      * If so, continue. This case also applies if the name is not specified and the existing record also has no name.
      * If not, give the user a warning that he/she is about to create a code list with the same identity but different names. If the user confirm, continue; otherwise, go back to the UI so that the user may change the name. This case also applies if the name is not specified and the existing record has a name.
  + If not, continue.
* Potential for unintentionally creating a classification scheme with identical name to existing records but different identities: Is there a Context\_Classification\_Scheme record with the same Scheme\_Name and Agency\_ID but with different Scheme\_ID? If so, give the user a warning “You are about to create a context classification scheme with an existing name but different identity (i.e., same Agency ID but different Scheme ID).” If the user confirm, continue; otherwise, go back to the UI.
* The scheme values cannot have duplicates within the context scheme.



Figure . Create a new context scheme UI

### Search and select a context classification scheme

A search UI should allow the user to look for an existing context scheme. A single search field should look through the Scheme\_ID, Name, Description, and Context\_Category\_ID (via the Name and Description columns of the Context\_Category table) columns of the Context\_Classification\_Scheme table.

A check box should be available so that the user can filter to only the latest version of the scheme.

The result dialog should show context category name, scheme name, scheme description, scheme version, and scheme agency name of each context scheme.

The UI should have an ‘OK’ button for the user to accept the selection; and, a ‘Cancel’ button which should return null. The ‘OK’ button should return a selected context scheme object or ID to the calling function.

### Update a context classification scheme

Invoke the Search and select a context classification scheme in 3.6.2. If null is returned, go back to the Application Home Screen. If a context scheme object or ID is returned, a UI same as in Figure 8 should be displayed. On clicking the ‘Save’ button, all the rules specified in 3.6.1 should be validated before committing the data to the database. In addition, the scheme value that is already used by a business context cannot be deleted. The application may lock those records when creating the UI or may implement this as a validation when saving the change.

The ‘Close’ button takes the user back to the Application Home Screen; though, a warning should be given that unsaved changes will be discarded and ask for a confirmation from the user.

### Discard a context classification scheme

Invoke the Search and select a context classification scheme in 3.6.2. If null is returned, go back to the Application Home Screen. If a context scheme object or ID is returned, check if there is any business context relying on the context scheme. If there is, show a dialog indicating that the context scheme cannot be discarded because there are business contexts depending on it – show some or all of the business contexts using the context scheme. Upon closing the dialog, take the user back to the Application Home Screen.

If there is no business context depending on the selected context scheme, ask the user for a confirmation. Once confirm, remove related records from the Context\_Scheme and Context\_Scheme\_Value tables. Upon completion, take the user back to the Application Home Screen.

## Code List Management

The code list management allows the user to manage data in the Code\_List and Code\_List\_Value tables.

### Create a new BIE code list without a base

A UI such as shown in Figure 9 should be displayed.



Figure . UI for Create a new BIE code list without a base

The ‘Save’ button creates new records in the Code\_List and Code\_List\_Value tables. Before creating the record, the following validation should be performed in addition to all required fields being specified.

* Potential for unintentionally creating a code list with the same identity but different names: Is there a Code\_List record with the same List\_ID and Agency\_ID?
  + If so, check if Version\_ID is the same as any of those existing records.
    - If so, give an error message that the same version cannot be created.
    - If not, check if Name is the same as the existing records.
      * If so, continue. This case also applies if the name is not specified and the existing record also has no name.
      * If not, give the user a warning that he/she is about to create a code list with the same identity but different names. If the user confirm, continue; otherwise, go back to the UI so that the user may change the name. This case also applies if the name is not specified and the existing record has a name.
  + If not, continue.
* Potential for unintentionally creating a code list with identical name but different identities: Is there a Code\_List record with the same Name and Agency\_ID but with different List\_ID? If so, give the user a warning “You are about to create a code list with the same name but different identity (i.e., same Agency ID but different List ID).” If the user confirm, continue; otherwise, go back to the UI.
* The Code column in the UI cannot have duplicates within the code list.

Note: The ‘Save’ button assigns the value ‘Editing’ to the Code\_List.State column.

Note: List\_ID, Agency\_ID, and VersionID columns are required. List\_ID is assumed to be unique within the Agency\_ID.

The following is the behavior of the ‘Publish’ button. Give a warning to the user that once published the code list cannot be modified but marked for deletion. If the user confirms, execute the same logic as in the ‘Save’ button logic above, except that the value ‘Published’ is assigned to the Code\_List.State column. If the user cancels, then go back to the UI.

The ‘Close’ button first warns the user that unsaved changes will be discarded. Upon confirmation returns the user to the Application Home Screen. The user can cancel and go back to the UI.

### Create a new BIE code list from another (by extension & restriction)

Invoke the Search and select a code list. If it returns null, go back to the Application Home Screen. If a code list object or ID is returned, verify that the Code\_List.Extensible\_Indicator = true, and Code\_List.State = ‘Published’ (only published code lists can be referenced). If not, show the error message indicating the reason and invoke the Search and select a code list again after the user acknowledges the error. If it is true, then show the UI such as shown in Figure 10.



Figure . UI for Create a BIE code list from another code list

At the initialization of the UI, the code list metadata is populated with content inherited from the base. The Code table in the UI is populated with the code values from the base as follows.

* Blue rows are where the Used\_Indicator =1 and Locked\_Indicator = 0. If the user unchecks (i.e., restricts) a blue row, the color changes to dull red. If the user rechecks, it changes back to blue.
* Bright red rows are where the Used\_Indicator = 0 and Locked\_Indicator = 0 or Locked\_Indicator = 1. The user cannot check or uncheck these rows.

Green rows are new code values added in this code list derivation. There is no green row at the initialization.

The ‘Save’ button write all data to the database with the Code\_List.State = ‘Editing’. The validation rules specified in 3.7.1 apply. The Used\_Indicator, Locked\_Indicator, Extension\_Indicator values should be written to the database as follows.

* Blue rows: Used\_Indicator = 1, Locked\_Indicator = 0, Extension\_Indicator = 0.
* Dull red rows: Used\_Indicator = 0, Locked\_Indicator = 0, Extension\_Indicator = 0.
* Bright red rows: Used\_Indicator = 0, Locked\_Indicator = 1, Extension\_Indicator = 0.
* Green rows: Used\_Indicator =1, Locked\_Indicator = 0, Extension\_Indicator = 1.

The ‘Publish’ button first gives a warning to the user that a published code list can no longer be changed but marked for deletion. If the user confirms, the same thing as in the ‘Save’ button happens, except the Code\_List.State = ‘Published’. If the user cancels, then go back.

The ‘Close’ button first warns the user that unsaved changes will be discarded. Upon confirmation returns the user to the Application Home Screen. If the user cancels, go back to the UI.

The ‘View’ button let the user open up the based code list so the user can see the original. This should open up the UI as in the View/Edit a BIE code list section but with all fields un-editable so that no modification can be made.

### Search and select a code list

A search UI should allow the user to search for a code list. A single search field should search through the Name, List\_ID, Agency\_ID, Version\_ID, Definition, and Remark columns. The user should be able to filter out code lists that are not extensible (Code\_List.Extensible\_Indicator = false) and code lists that are already marked for deletion (Code\_list.State = ‘Deleted’). The user can select one of the matched code lists and click an ‘OK’ button, then the UI returns the selected code list object or ID. A ‘Cancel’ button returns null. In both cases, go back to the calling dialogue.

### View/Edit a BIE code list

Invoke the Search and select a code list. If it returns null, go back to the Application Home Screen. If a code list object or ID is returned, show the UI similar to that in Figure 9 or Figure 10, if it is a derived code list. However, the buttons should be enabled or disabled based on the code list’s state. If the state is Editing, then all the buttons are enabled. If the state is Published, only the Close button is enabled. Similar logic in 3.7.1 and 3.7.2 apply.

### Discard a BIE code list

Invoke the Search and select a code list. If it returns null, go back to the Application Home Screen. If a code list object or ID is returned, check that the code list’s state is Editing. If so, remove related records from Code\_List and Code\_List\_Value tables. Upon the successful removal, go back to the Application Home Screen.

If the state is not Editing, show an error message “The selected code list cannot be discarded because it is in the Deleted or Published state. Use the Delete Code List menu instead (to mark the published code list as deleted).” Maybe also show the code list’s state in the error message. Upon acknowledging the error message, go back to the code list search page.

### Delete a BIE code list

Invoke the Search and select a code list. If it returns null, go back to the Application Home Screen. If a code list object or ID is returned, check that the code list’s state is Published. If so, change the State of the code list in the database to Deleted. Upon the successful removal, go back to the Application Home Screen.

If the state is not Published, show an error message “The selected code list cannot be marked for deletion because it is not in the Published state. Or the code list is already in the Deleted state (and it can be purged by a database admin). ” Maybe also show the code list’s state in the error message. Upon acknowledging the error message, go back to the code list search page.

## Generate a standalone OAGIS Expression for a top-level ABIE

A UI similar to Figure 11 should be displayed.



Figure . UI for the standalone BOD generation

The ‘Add’ button invokes the top-level ABIE search and select – see 3.3.7. However, multiple selections of the top-level ABIEs must be allowed in this case.

The user can select the types of information to generate in the annotation. The ‘Inherit if empty?’ option is only enabled if the ‘BIE Definition’ option is selected. If enabled and selected by the user, it means that if the BIE definition is empty, the BIE definition should be populated with the definition from the associated CC in the generated schema.

A specific expression must be selected, i.e., one expression can be selected at a time.

The ‘Put all schemas in the same package’ option indicates whether to put all the standalone BODs into a single *xsd:schema* element.

The ‘Generate’ button returns a zip or XSD file containing the generated standalone BOD expressions. See each subsection below for specific generation logic.

Let us assume that an object instance called annotationOption contains information about the annotation to generate as selected by the user.

### Generate a standalone XML Schema

Generate the xsd:schema element with the following attributes

|  |  |
| --- | --- |
| **Attribute Name** | **Values** |
| xmlns:xsd | http://www.w3.org/2001/XMLSchema |
| xmlns | http://www.openapplications.org/oagis/10 |
| xmlns:xml | http://www.w3.org/XML/1998/namespace |
| targetNamespace | http://www.openapplications.org/oagis/10 |
| elementFormDefault | qualified |
| attributeFormDefault | Unqualified |

If the Schema Package flag is true, generate only one xsd:schema element and put all the top-level ABIEs under the same xsd:schema element in a single file. Otherwise, put each top-level ABIE in each xsd:schema element in an individual file.

Let’s assume that the xsd:schema element is in the XML DOM Document object named tlABIEDOM. The xsd:schema element is represented by a variable schemaNode.

-------------------------------------------------------------------

Create a vector of GUIDs to store GUIDs of code lists and agency ID lists already generated. Let us name this vector guidVector. This vector will store Code List BDT GUIDs. It is either globally accessible during the generation or may be passed as an additional argument in all the generation functions below.

See 3.1.1.8 for string functions used in the rest of the section. In addition, some objects and functions correspond to the Xerces Java DOM API. The dot expressions (e.g., gABIE.Definition, aNode.getNodeValue( )) used refer to a child object, an object function, a child table, or a child table column. In some cases, the child table and the child column should be converted to a corresponding object in the implementation. In other cases, only the value is needed in the implementation for the child table column. Interpretation of these expressions should be intuitive from the flow.

#### GenerateTopLevelABIE(tlASBIEP, tlABIEDOM)

Description: Generate a root element node corresponding to the given tlASBIEP and its content.

Argument: tlASBIEP = the top-level ASBIEP to be generated, tlABIEDOM = an XML DOM Document object

Return: Updated tlABIEDOM.

generateBIEs(aABIE, rootSeqNode)

rootSeqNode = generateABIE(aABIE, rootEleNode)

Return tlABIEDOM

rootENode = GenerateTopLevelASBIEP(tlASBIEP, schemaNode)

aABIE = QueryTargetABIE(tlASBIEP)

#### GenerateTopLevelASBIEP(gTlASBIEP, gSchemaNode)

Description: Add a child root xsd:element node with the following attributes to the gSchemaNode.

Argument: gTlASBIEP = the given top-level ASBIEP (the BOD), gSchemaNode = XML DOM Node object representing an xsd:schema element.

Return: rootENode = the xsd:element node added to the gDOM.

Create a DOM Element node, rootEleNode, as a child of the gSchemaNode. Create DOM Attribute node or nodes as children of the created Element node with the follow node name and node value.

|  |  |
| --- | --- |
| **Attribute Node Name** | **Node Value** |
| Name | UpperCamelCase(First(gTlASBIEP.Based\_ASCCP\_ID.DEN)) |

Call the function GenerateTopLevelASBIEPAnno(gTlASBIEP, rootENode)

Return the rootEleNode.

#### GenerateTopLevelASBIEPAnno(gTlASBIEP, gRootENode)

Description: Generate annotation for the gTlASBIEP per the annotation option specified by the user.

Argument: gTlASBIEP = the given top-level ASBIEP, for which the annotation will be generated, gRootENode = the DOM Element Node representing the root xsd:element (the BOD’s root xsd:element).

Return: The DOM Element Node object representing the xsd:annotation that has been created.

IsAnnoOptionEmpty( )

Return Null

Y

#### IsAnnoOptionEmpty( )

Description: This function checks to see if the user has selected any annotation to be generated. If none has been selected return true; otherwise, return false.

Return: Boolean

Check whether the annotationOption is empty or that all of the option is false. If so, return true. Otherwise, return false.

#### GenerateABIE(gABIE, gASBIEPElementNode)

Description: Add a child xsd:complexType/xsd:sequence element node to the gElementNode.

Argument: gABIE = the given ABIE corresponding to the xsd:complexType, gASBIEPElementNode = a DOM Element node object representing the xsd:element corresponding to the ASBIEP of the gABIE.

Return: a DOM Element node representing the xsd:sequence element.

Add an xsd:complexType element, namely CTypeNode, as a child of the gElementNode. Call GenerateABIEsAnno(gABIE, CTypeNode) to add necessary annotation as a child of the xsd:complexType element. Then add an xsd:sequence element as a child of the xsd:complexType. Return the DOM Element node of the xsd:sequence.

#### GenerateABIEsAnno(gABIE, gABIECTypeNode)

#### GenerateBIEs(gABIE, gPNode, gSchemaNode)

Description: Recursively generate the content of the given gABIE which represents an XSD complex type.

Argument: gABIE = the ABIE for which the content will be generated, gPNode = xsd:sequence DOM Element node, parent node of the gABIE’s content, gSchemaNode = XML DOM Node object representing a root xsd:schema element.

Return: The updated gSchemaNode.

GenerateBBIE(childBIEs[i], aBDT, gNode, gSchemaNode)

aBDT = QueryAssocBDT(childBIEs[i])

Return gSchemaNode

Y

childBIEs[i] is an ASBIE?

node = GenerateASBIE(childBIEs[i], gPNode )

Y

childBIEs[ ] = QueryChildBIEs(gABIE)

node = GenerateASBIEP(anASBIEP, node)

anABIE = QueryTargetABIE(anASBIEP)

N

node = GenerateABIE(anABIE, node)

node = GenerateBIEs(anABIE, node, gSchemaNode)

anASBIEP **=** QueryAssocToASBIEP(childBIEs[i])

Next childBIE[i]?

N

Append xsd:any element to the gPNode

gABIE.ACC.DEN = “Any User Area. Details”

Y

N

#### QueryTargetABIE(gASBIEP)

Description: Retrieve from the database the ABIE, to which the gASBIEP points.

Argument: gASBIEP = the ASBIEP, from which the target ABIE is to be retrieved.

Return: An ABIE object

Return an ABIE object constructed from the foreign key ASBIEP.Role\_Of\_ABIE\_ID.

#### QueryChildBIEs(gABIE)

Description: Retrieve from the database ASBIE and BBIE children of the gABIE, whose max cardinality is not zero.

Argument: gABIE = the ABIE, from which the retrieved ASBIEs and BBIEs are associated.

Return: An array or a vector object containing the ASBIE and BBIE children.

Query the database for ASBIE and BBIE records whose the ASBIE.Assoc\_From\_ABIE\_ID = gABIE.ABIE\_ID or BBIE.Assoc\_From\_ABIE = gABIE.ABIE\_ID. Discard the ASBIE and BBIE where its Max\_Cardinality = 0. Order the ASBIE and BBIE objects/records in the array or vector by the ASBIE.Sequencing\_Key and BBIE.Sequencing\_Key.

#### GenerateASBIE(gASBIE, gPNode)

Description: Create a DOM Element node with the cardinality attributes corresponding to the information in the gASBIE.

Argument: gASBIE = the given ASIBE, gPNode = XML DOM Node object which will be the parent node of the gASBIE’s content.

Return: The created DOM Element node for the gASBIE.

Create an XML DOM element node as a child of the gPNode with the following child DOM Attribute nodes.

|  |  |
| --- | --- |
| **Attribute Node Name** | **Node Value** |
| minOccurs | gASBIE.Cardinality\_Min |
| maxOccurs | gASBIE.Cardinality\_Max. If it is -1, then the value is “unbounded”. |
| nillable | If gASBIE.Nillable is null, don’t generate the Attribute node. Otherwise the value is gASBIE.Nillable. |

#### QueryAssocToASBIEP(gASBIE)

Description: Get the ASBIEP, to which gASBIE points.

Argument: gASBIE = the given ASBIE, from which to retrieve the ASBIEP.

Return: The retrieved ASBIEP.

Get an ASBIEP record from the ASBIEP table where gASBIE.Assoc\_To\_ASBIEP\_ID = ASBIEP.ASBIEP\_ID.

#### GenerateASBIEP(gASBIEP, gElementNode)

Description: Add a child DOM Attribute node to the gElementNode representing the element’s name retrieved from the gASBIEP.

Argument: gASBIEP = the given ASBIEP from which the element name will be extracted, gElementNode = the given XML DOM Element node object.

Return: The updated gElementNode.

Add the following child DOM Attribute node to the gElementNode.

|  |  |
| --- | --- |
| **Attribute Node Name** | **Node Value** |
| name | UpperCamelCase(First(gASBIEP.DEN)) |

#### QueryTargetABIE(gASBIEP)

Description: Retrieve the ABIE the gASBIEP qualifies.

Argument: gASBIEP = the given ASBIEP, from which an ABIE is retrieved.

Return: The retrieved ABIE object.

Get the ABIE record that matches gASBIEP.Role\_Of\_ABIE\_ID = ABIE.ABIE\_ID.

#### QueryAssocBDT(gBBIE)

Description: Get the BDT used by the gBBIE.

Argument: gBBIE = the given BBIE, for which the BDT will be retrieved.

Return: The BDT.

Return aBDT

Get the BCC record where gBBIE.Based\_BCC\_ID = BCC.BCC\_ID

Get the BDT record from the DT table where BCCP.BDT\_ID = DT.DT\_ID, assign it as the aBDT object

Get the BCCP record where BCC.Assoc\_To\_BCCP\_ID = BCCP.BCCP\_ID

#### GenerateBBIE(gBBIE, gBDT, gPNode, gSchemaNode)

Description: Create a BBIE xsd:element node, its type, its attributes and if necessary code list and agency ID list.

Argument: gBBIE = the given BBIE, for which the xsd:element node is to be created, gBDT = the BDT representing the content of the BBIE, gPNode = a parent (xsd:sequence) DOM Element node of the BBIE, gSchemaNode = the root xsd:schema DOM Element node.

Return: The generated BBIE xsd:element node

Create a DOM Element node, eNode, representing the xsd:element, as a child of the gPNode.

SCs[ ].size = 0?

Create a DOM Attribute node, nameANode, representing the name attribute of the eNode. Set the value to GetBBIEElementName(gBBIE). Also handle the Min and Max Cardinality attributes here.

**(1)** Handle nillable, default and fixed

SCs[ ] = GetBBIESCs(gBBIE)

N

clNode = GenerateCodeList(

aCL, gBDT, gSchemaNode)

**(2)** Get a code\_list object, aCL, assigned to the gBBIE if any.

N

**(7)** Add an XML DOM Attribute node, tNode, representing the ‘type’ attribute as a child of the eNode and set its value.

eNode = GenerateSCs(gBBIE, eNode, SCs[ ], gSchemaNode)

**(3)** Add an XML DOM Attribute Node, tNode, representing the ‘type’ attribute as a child of the eNode and set its value.

**(6)** Create a DOM Element node structure for the xsd:extension, extNode, under the eNode.

**(7)** Add the ‘base’ DOM Attribute Node as a child of the extNode, and set its value.

Is aCL null?

Is gBBIE.

BDT\_Primitive\_Restriction\_ID

null?

Y

**(4)** Add an XML DOM Attribute Node, tNode, representing the ‘type’ attribute as a child of the eNode and set its value.

SCs[ ].size = 0?

Y

Y

Return

eNode

Create the following DOM Element Node structure as a child of the eNode xsd:complexType/xsd:simpleContent/

xsd:extension. Let extNode denote the xsd:extension.

**(5)** Add the ‘base’ DOM Attribute Node as a child of the extNode and set its value.

SCs[ ].size = 0?

N

Y

N

N

Y

isCodeListGenerated(aCL)

N

Y

Return

eNode

Is BBIE an attribute?

Create a DOM Element node, eNode, representing the xsd:attribute as a sibbling of the gPNode.

Create a DOM Attribute node, nameANode, representing the name attribute of the eNode. Set the value to GetBBIEAttributeName(gBBIE). Also handle cardinality using ‘optional’ attribute node here.

Y

N

1. Add the following logic to deal with nillable, default, and fixed value.
   1. If gBBIE.isNillable = True, add a child DOM Attribute node to eNode with node name, “nillable”, and node value, “true”. Ignore this flag in case the BBIE is an attribute and log a warning message that this flag is not compatible for the XML schema generation.
   2. If gBBIE.Default != null, add a child DOM Attribute node to eNode with node name, “default”, and node value, gBBIE.Default.
   3. If gBBIE.Fixed\_Value != null, add a child DOM Attribute node to eNode with node name, “fixed”, and node value, gBBIE.Fixed\_Value.
   4. If both gBBIE.Default and gBBIE.Fixed\_Value is not null, throw an error. The validation rule should have caught this during the BIE editing.

Note that XSD cannot express the isNull option in the BBIE table (i.e., fix the value to Null), so there is nothing to do for that.

In the case that gBBIE is an attribute, it is expected none of the blocks after this block is applied except either block (3), (4), or (7).

1. Let aCL be a code list object. Follow the following sequence to get a code list object. Get aCL from gBBIE.Code\_List\_ID, if it is not null. Or get aCL from gBBIE.BDT\_Primitive\_Restriction.Code\_List\_ID, if it is not null. Otherwise, only if gBBIE.Code\_List\_ID and gBBIE.BDT\_Primitive\_Restriction\_ID are null, get aCL from gBDT.BDT\_Primitive\_Restriction[isDefault = True].Code\_List\_ID. If all cases are null, then aCL is null. This and the next few blocks in the diagram deal with the case where the user assigns a code list at the BIE level or that the default BDT primitive restriction is a code list. This means that if the default BDT primitive restriction is a specific code list, the user can only assign another (more restrictive) code list at the BIE level, i.e., the BBIE.BDT\_Primitive\_Restriction\_ID column cannot be used.
2. Set the tNode value to gBDT. BDT\_Primitive\_Restriction[isDefault = True].CDT\_Primitive\_Expression\_Type\_Map.  
   XSD\_BuiltIn\_Type.BuiltIn\_Type.
3. Set its value to gBBIE.BDT\_Primitive\_Restriction.CDT\_Primitive\_Expression\_Type\_Map.XSD\_BuiltIn\_Type.BuiltIn\_Type.
4. If gBBIE.BDT\_Primitive\_Restriction\_ID is null, set the node value to the BuiltIn\_Type the same way as box (3) above. Otherwise set the node value to the same as in box (4) above.
5. The node structure is xsd:complexType/xsd:simpleContent/xsd:extension. Let extNode denote the xsd:extension.
6. Set the value to getCodeListTypeName(aCL).

#### GetBBIEElementName(gBBIE)

Description: Return a string representing the element name of the gBBIE.

Argument: gBBIE = the given BBIE for which the element name will be generated.

Return: A string.

Trace the gBBIE down through BBIEP and then BCCP. Return UpperCamelCase(BCCP.Property\_Term).

#### GetBBIEAttributeName(gBBIE)

Description: Return a string representing the attribute name of the gBBIE, in the case that the BBIE is a schema attribute.

Argument: gBBIE = the given BBIE for which the attribute name will be generated.

Return: A string.

Trace the gBBIE down through BBIEP and then BCCP. Return LowerCamelCase(BCCP.Property\_Term).

#### QueryBBIESCs(gBBIE)

Description: Return records/objects in the BBIE\_SC table relevant to gBBIE.

Argument: gBBIE = the given BBIE from which BBIE\_SC objects are to be retrieved.

Return: An array of the SC objects. The array may be empty.

Get BBIE\_SC records whose BBIE\_ID matches gBBIE.BBIE\_ID.

#### isCodeListGenerated(gCL)

Description: Return Boolean indicating whether the code list type has already been generated, true if generated and false otherwise.

Argument: gCL = the given code list object to be investigated.

Return: Boolean

Check whether the gCL.Code\_List\_GUID is already in the guidVector. If so return true, otherwise return false.

#### GetCodeListTypeName(gCL)

Description: Return the type name of the gCL object.

Argument: gCL = the given code list object, for which the type name to be constructed.

Return: A string representing the type name of the gCL.

Return { gCL.Name + (gCL.Name.endsWith(“Code”)? “” : “Code”) + “ContentType” + “\_” + gCL.Agency\_ID + “\_” + gCL.List\_ID + “\_” + gCL.Version\_ID}.

#### GenerateCodeList(gCL, gBDT | gSC, gSchemaNode)

Description: Generate a type definition for the gCL. This function may become two physical functions in the implementation – one generating the code list related to the gBDT and another related to gSC.

Argument: gCL = the given code list, for which the type definition will be generated, gBDT = the BDT related to the code list, gSC = the SC related to the code list, gSchemaNode = the root xsd:schema DOM Element node.

Return: The generated DOM XML Element node representing the xsd:simpleType corresponding to the gCL.

Create a DOM Element node, stNode, representing the xsd:simpleType, as a child of the gSchemaNode.

Add a DOM Element node, rtNode, representing the xsd:restriction, as a child of the stNode.

Get all Code\_List\_Value records belonging to gCL where the Used\_Indicator = True. Put them in the gCLVs[ ].

For each gCLVs[i]

**(1)** Add a DOM Attribute node, stNameNode, with node name = “name”, as a child of the stNode and set its value.

**(3)** Add a DOM Attribute node, with node name = “base”, as a child of the rtNode, and set its value.

Create an xsd:enumeration DOM Element node and its child ‘value’DOM Attribute node. Set the Attribute node value to gCLVs[i].Value.

More gCLVs[i]?

Return stNode

Y

N

**(2)** Add a DOM Attribute node, stIdNode, with node name = “id”, as a child of the stNode and set its value.

1. Set its value to GetCodeListTypeName(gCL).
2. Set its value to gCL.Code\_List\_GUID.
3. This box sets the base type of the code list to XSD type associated with the default BDT primitive restriction or the default BDT SC primitive restriction in the model. In other words, it is assumed that the code values have to be compliance with the default XSD type.
   1. In the case of gBDT is given, set the value as follows. Let dPrim = gBDT.BDT\_Primitive\_Restriction  
      [isDefault = True]. Set the node value to { dPrim.Code\_List\_ID != null ? “xsd:token” : dPrim.CDT\_Primitive\_Expression\_Type\_Map.XSD\_BuiltIn\_Type.BuiltIn\_Type }
   2. In case of the gSC is given, set the value as follows. Let dPrim = gSC.BDT\_SC\_Primitive\_Restriction  
      [isDefault = True]. Set the node value to { dPrim.Code\_List\_ID != null ? “xsd:token” : dPrim.CDT\_SC\_Primitive\_Expression\_Type\_Map.XSD\_BuiltIn\_Type.BuiltIn\_Type }

#### GenerateSCs(gBBIE, gBBIENode, gSCs[ ], gSchemaNode)

Description: Generate xsd:attribute DOM Element nodes corresponding to the gSCs[ ].

Argument: gBBIE = BBIE object owning the gSCs[ ], gBBIENode = the parent (ancestor) xsd:element DOM Element node of the xsd:attribute to be generated, gSCs[ ] = an array of supplementary component objects, for which the xsd:attribute will be generated, gSchemaNode = the root xsd:schema DOM Element node.

Return: The updated gBBIENode.

**(1)** Get the descendant xsd:complexType/ xsd:extension of the gBBBIENode, denote it by tNode.

For each gSCs[i]

Generate a DOM Element node, aNode, for the xsd:attribute element, as a child of the tNode.

**(2)** Generate a DOM Attribute node, aNameNode, for the ‘name’ attribute of the xsd:attribute element. And set its value.

**(7)** Handle gSCs[i]’s default and fixed value options.

**(3)** Get a code list object, aCL, assigned to the gSCs[i], if any.

Generate a DOM Attribute node, aTypeNode, for the ‘type’ attribute as a child of aNode and set its value to GetCodeListTypeName(aCL).

aCL = null?

isCodeListGenerated(aCL)?

GenerateCodeList(aCL, gSCs[i], gSchemaNode)

N

N

**(5)** Generate a DOM Attribute node, aTypeNode, for the ‘type’ attribute as a child of aNode and set its value.

primRestriction = null?

primRestriction = gSCs[i]. DT\_SC\_Primitive\_Rescrition\_ID

**(4)** Get an agency id list object, anAL, assigned to the gSCs[i], if any.

aAL = null?

isAgencyListGenerated(aAL)?

GenerateAgencyList(aAL, gSCs[i], gSchemaNode)

N

N

Y

Generate a DOM Attribute node, aTypeNode, for the ‘type’ attribute as a child of aNode and set its value to GetAgencyListTypeName(aAL).

Y

**(6)** Generate a DOM Attribute node, aTypeNode, for the ‘type’ attribute as a child of aNode and set its value.

Return gBBIENode

N

Y

Y

More gSCs[i]?

N

Y

Y

1. The child Element node should be either an xsd:simpleType or xsd:complexType element.
2. In other words, aNameNode’s name is ‘name’. Set the value to:
   1. If the DT\_SC.Representation\_Term is Text, LowerCamelCase(gSCs[i].DT\_SC.Property\_Term)
   2. If the DT\_SC.Representation\_Term is Identifier, concat(LowerCamelCase(gSCs[i].DT\_SC.Property\_Term), “ID”).
   3. Else, concat(LowerCamelCase(gSCs[i].DT\_SC.Property\_Term), UpperCamelCase(gSCs[i].DT\_SC.Representation\_Term)).
3. Let aCL be a code list object. Get aCL in the following order. Get aCL from gSCs[i].Code\_List\_ID, if it is not null. Or get it from gSCs[i]. DT\_SC\_Primitive\_Restriction\_ID.Code\_List\_ID, if it is not null. Else and only if both gSCs[i].Code\_List\_ID and gSCs[i].DT\_SC\_Primitive\_Restriction\_ID are null, try getting it from gSCs[i].DT\_SC.BDT\_SC\_Primitive\_Restriction[isDefault = True].Code\_List\_ID. If all the cases return null, then aCL is null. See the note in section “3.8.1.21 GenerateCodeList(gCL, gBDT | gSC, gSchemaNode)” to how the code list and primitive restriction are dealt with.
4. Let aAL be an agency ID list object. Get aAL from gSCs[i].Agency\_ID\_List\_ID, if it is not null. Or get it from gSCs[i].DT\_SC\_Primitive\_Restriction\_ID.Agency\_ID\_List\_ID, if it is not null. Else and only if both gSCs[i].Agency\_ID\_List\_ID and gSCs[i].DT\_SC\_Primitive\_Restriction\_ID are null, try getting it from gSCs[i].DT\_SC.BDT\_SC\_Primitive\_Restriction[isDefault = True]..Agency\_ID\_List\_ID. If all the cases return null, then aAL is null.
5. Set the value of aTypeNode as gSCs[i].DT\_SC.BDT\_SC\_Primitive\_Restriction[isDefault = True]. CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map.XSD\_BuiltIn\_Type.BuiltIn\_Type.
6. Set the value of aTypeNode as gSCs[i].DT\_SC\_Primitive\_Restriction\_ID.CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map.XSD\_BuiltIn\_Type.BuiltIn\_Type.
7. Handle the default and fixed value options of gSC[i] as follows.
   1. If gSCs[i].Default != null, add DOM Attribute node as a child of the tNode. Set its value to gSCs[i].Default.
   2. If gSCs[i].Fixed\_Value != null, add DOM Attribute node as a child of the tNode. Set its value to gSCs[i].Fixed\_Value.
   3. If both gSCs[i].Default and gSCs[i].Fixed\_Value are not null, throw an error.

#### isAgencyListGenerated(gAL)

Description: Return Boolean indicating whether the agency ID list has already been generated, true if generated and false otherwise.

Argument: gAL = the given agency ID list object.

Return: Boolean

Check whether the gAL.Agency\_ID\_List\_GUID already in the guidVector. If so return true, otherwise return false.

#### GetAgencyListTypeName(gAL)

Description: Return the type name of the gAL object.

Argument: gAL = the given agency ID list object, for which the type name to be constructed.

Return: A string representing the type name of the gAL.

Return { “clm” + gAL.Agency\_ID + gAL.List\_ID + gAL.Version + “\_” + UpperCamelCase(gAL.Name) + “ContentType” }

#### GenerateAgencyList(gAL, gSC, gSchemaNode)

Description: Generate the Identification Content Type corresponding to the gAL as a child of the xsd:schema element

Argument: gAL = the given agency ID List of object, for which the type to be generated, gSC = the supplementary component object that uses gAL, gSchemaNode = the xsd:schema DOM Element node – parent of the type node to be generated.

Return: The generated DOM Element node representing the xsd:simpleType corresponding to the gAL.

Create a DOM Element node, stNode, representing the xsd:simpleType, as a child of the gSchemaNode.

Add a DOM Element node, rtNode, representing the xsd:restriction, as a child of the stNode

Get all Agency\_List\_Value records belonging to gAL. Put them in the gALVs[ ].

For each gALVs[i]

**(1)** Add a DOM Attribute node, stNameNode, with node name = “name”, as a child of the stNode and set its value.

Add a DOM Attribute node, with node name = “base”. Set the node value to “xsd:token”.

Create an xsd:enumeration DOM Element node and its child ‘value’DOM Attribute node. Set the Attribute node value to gALVs[i].Value.

More gALVs[i]?

Return StNode

Y

N

**(2)** Add a DOM Attribute node, stIdNode, with node name = “id”, as a child of the stNode and set its value.

1. Set its value to GetAgencyListTypeName(gAL).
2. Set its value to gAL.Agency\_ID\_List\_GUID.

## Namespace Management

A UI should be available for the user to Create and Update a Namespace. A namespace can only be deleted when there is no reference. Only the user who is the owner as indcated by the Owner\_User\_ID column can update or delete the namespace. The user who created the record is the first owner. The owner can assign another user as an owner. Hence, there must be another functionality for the owner to assign a new owner. Changes to the namespace are not tracked with history records. The WHO columns must be properly updated as the record gets created and updated. There is no other specific logic to namespace management functionalities.

## Manage CCs and DTs

Note: See specifics for management of User Extension Group ACC in 3.3.9.

### Architecture for managing revisions

#### Current and history records

Logically, CCs revisions and releases are managed in two sets of records – current and history. This can be viewed as two sets of tables as shown in Figure 12 below. However, both current and history records are implemented with only a single set of tables; and the current and history records are differentiated by the revision\_num column. The current record always have a revision\_num of 0, while the history record has a revision\_num of 1 or higher. The history record always points to its corresponding current (master) record.



Figure : Concepts of current and history records

Everytime there is a change to a CC, a history record is inserted. For a new revision, the revision\_num is incremented from the latest revision. For every change during the revision process, the history records are inserted with the same revision\_num but differing revision\_tracking\_num. The revision\_tracking\_num supports a machinsm for the user to undo his/her changes. CCs have a state which can be Editing, Candidate, Published, and Deprecated. Once a revision of a CC is published, the history record whose revision\_tracking\_num is 1 or more is deleted except the highest (latest) one to reflect the final picture of the CC’s revision.

The history record also has a revision\_action column, which indicates the database transaction that has happened to the current record. The value can semantically means insert, update, and delete. The purpose of this column is two-fold. First, it supports undoing changes during the revision. The other is to support the release of the CC content. This is particularly important when the action is ‘delete’. When an association is deleted, it is no longer present in the current record. Therefore, the history record indicates that the association is no longer exist in releases after the deletion has occurred (this works in conjunction with the release\_id column).

CC tables have a common set of columns, such as the revision\_num and others discussed above, to keep track of changes, revision, and release. See the data model for the definition of these columns and how they should be populated in general. Unless otherwise specifically specified in other subsections of section 3.10, explanations in the data model and in this 3.10.1 section shall be the guideline for populating these revision related columns.

#### CC states

All CC entities have a State column that keeps track of its life-cycle. The state can be Editing, Candidate, Published, and Deprecated. An ASCC and BCC can also have a Deleted state, but it is not represented by the State column.

Editing state means that the entity is being edited. At a particular point in time, an entity can be edited by only one user designated by the entity’s owner\_user\_id column. The current owner can assign another another user as owner to replace him/herself. The Editing state facilitates a work-in-progress situation. That is, while an entity is in the Editing state only the owner can see the entity.

The owner can move the entity from the Editing to the Candidate state. The Candidate state facilitates a draft review. This is when the entity can be visible and reviewed by other users. Comments may be given by other users. The owner is not allowed to make changes to an entity in the Candidate state; however, he/she can move the entity back to the Editing state to make changes.

From the Candidate state, the owner can move the entity to the Published state. Entities in Published states are stable and reusable by other entities. Once the owner move an entity into the Published state, he has relingquished his control and another user can create a new revision of that entity.

An entity may be deprecated after published, then it would have a Deprecated state. Deprecated entities must be restricted from reuse. In other words, no new references to the deprecated entity shall be made. The State 4 column in Table 2 shows an example that the Address ACC happens to be deprecated during the editing of the Broker Party ACC. In such situation, the owner of the Broker Party ACC shall be notified.

Dependencies between entities means that there is a restriction on the states of dependent entities. Generally, the referring entity shall not have a state that is more advanced than the referred to entity. For example, assuming a user is building a new Broker Party ACC. It has an ASCC that uses a new ASCCP Office Address which in turn reuses the Address ACC. During the process of building the Broker Party ACC, Table 2 below displays states of these entities at different points in time. The columns, State 3 and State 5, illustrate invalid situation. In State 3, the Broker Party ASCCP has a Candidate state, which is more advanced than the Editing state of the Broker Party ACC it is referring to. In such situation, when reviewing the Broker Party ASCCP, users cannot see the content of the Broker Party ACC. In State 5, the Broker Party ASCCP is in a Published state, which is again more advanced than the Candidate state of the Broker Party ACC. In this case, the Broker Party ASCCP cannot really in a reusable state, because its actual data structure, defined in the Broker Party ACC, may change. State 5 column has another problem in that the state associated with the Broker Party. Office. Address ASCC has more advanced state than that of the Office Address ASCCP. In this case, users cannot review the Broker Party ACC that is in the Candidate state because the content it is dependent on, the Office Address ASCCP, is not accessible.

Since an entity can shuttle back and forth between Editing and Candidate state, this presents another requirement. When switching an entity state from Candidate back to Editing, the system has to check if there is a referring entity that would be having a more advanced state after the switch back.

An exception to the above rule has to do with the Deprecated state. The referring entity can be deprecated before the referred to entity, given that both entities are in the Published state before the referring entity is deprecated. For example, an ASCCP can be deprecated while the ACC it refers to is not. Similarly, an assocition can be deprecated while the ASCCP or BCCP it refers to is not.

Table : Example CC states dependency

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Entity | State 1 | State 2 | State 3 | State 4 | State 5 | State 6 |
| Broker Party ASCCP | Editing | Editing | *Candidate* | Editing | *Published* | Editing |
| Broker Party ACC | Editing | Editing | Editing | Candidate | Candidate | Published |
| Broker Party. Office. Address ASCC | Editing | Editing | Editing | Candidate | *Candidate* | Published |
| Office Address ASCCP | Editing | Published | Editing | Published | Editing | Published |
| Address ACC | Published | Published | Published | Deprecated | Published | Published |
| Address. Address Line. Address Line BCC | Published | Published | Published | Published | Published | Published |
| Address Line. Seq Text BCCP | Published | Published | Published | Published | Published | Published |
| Seq Text. Type BDT | Published | Published | Published | Published | Published | Published |
| Address. Geo Cordinate. Coordinate ASCC | Published | Published | Published | Deprecated | Published | Published |
| Geo Coordinate. Coordinate ASCCP | Published | Published | Published | Deprecated | Published | Published |
| Coordinate ACC | Published | Editing | Candidate | Deprecated | Published | Published |

An ASCC and BCC can additionally have a Deleted state. No other kinds of entities cannot have a Deleted state (once created, they cannot be deleted). Deleted state is not represented by the State column but a Revision\_Action column. Other kinds of entities are implicitly deleted when they are not used in any association. An ASCC or BCC can only be transitioned into the Deleted state if it is in the Published or Deprecated state.

#### Summary of state related business rules

|  |  |
| --- | --- |
|  | Referring entity cannot have a more advance state than the referred to entity, with one exception. The referring entity can be deprecated before the referred to entity, given that both entities are in the Published state before the referring entity is deprecated. For example, an ASCCP can be deprecated while the ACC it refers to is not. Similarly, an assocition can be deprecated while the ASCCP or BCCP it refers to is not. |
|  | Only the owner can make changes to the entity. |
|  | There can be only one active revision of an entity at a particular point in time. I.e., there cannot be two revisions of an entity in Editing or Candidate state. I.e., A new revision can only be created on an entity in the Published state or Deprecated state. |
|  | A new revision cannot be created on a deprecated entity. Does this make sense??? |
|  | No new reference to a deprecated entity shall be allowed. I.e., a deprecated entity shall no longer be reused. |
|  |  |

#### Unit of control and concurrency management

Because CCs are typically interconnected, this section describes the method to manage the concurrency of accessing entities. The explanation about CC states in the previous section provides pre-requisite for concurrency control.

For concurrency control, units of control (or unit of check out) must be first defined. They are as follows:

1. ACC Unit of Control (UoC): States of an ACC and its child ASCC and BCC are always checked out altogether. This means that their states shall be transitioned together.
2. ASCCP UoC: An ASCCP can be checked out independently.
3. BCCP UoC: An BCCP can be checked out independently.
4. BDT UoC: A BDT can be checked out independently.

Owners of all active UoCs (UoCs in the Editing state) must be notified of a state change event associated with these UoCs. A mechanism may be created to filter events such that only the owners of UoCs that are directly and indirectly referring and referred to by the UoC generating the event are notified.



Figure : Example core components

Taking the core components in Figure 13 as an example, let’s assume that the PartyScreenResponseType ACC is first checked out into the Editing state by a user – U1. Then, another user, U2, checked out the CoordidateType ACC into the Editing state. U1 shall be notified of that event.

Next, U1 transitioned the PartyScreenResponseType ACC into the Candidate state. U2 shall be notified of that event. U2 should review the PartyScreenResponseType ACC to ensure that the changes do not overlap with the changes he was going to make to the CoordinateType ACC.

Next, U2 moved the CoordinateType ACC into the Candidate state. Similarly, U1 shall be notified of the event and review the content in the CoordinateType ACC to ensure that there would be no overlap with the changes he is going to make to the PartyScreenRespsonseType ACC.

#### Release management

This is a future version feature.

### CCs and DTs management home screen

The CCs and DTs management home screen is where all the CCs and DTs management functionalities can invoked. This includes 3.10.4 View CCs, 3.10.5 Create a new ACC (and an ASCCP), 3.10.6 Create a new revision of an ACC, 3.10.7 Edit an ACC, 3.10.13 Create a new ASCCP, 3.10.14 Create a new revision of an ASCCP, 3.10.15 Create a new BCCP, and 3.10.16 Create a new revision of a BCCP.

### ACC and ASCCP content rendering

The tree model of the ACC with inheritance maybe rendered such like Figure X below. The figure illustrates the rendering of the X ASCCP, which uses the X ACC, where the X ACC is based on the Y ACC, and the Y ACC is based on the Z ACC. Each of the ACCs has two children ASCCPs. In addition, the Y1 ASCCP uses the A ACC, but the A ACC is based on the B ACC. Basically, the ACC in the bottom box is the ACC used by the ASCCP on the left.

Note that the Y1 ASCCP may be a semantic group, in which case it shall be rendered in a different shape.

X ACC

Y ACC

Z ACC

X ASCCP

Z1 BCCP

Z2 BCCP

Y1 ASCCP

Y2 BCCP

X1 BCCP

X2 BCCP

A ACC

A1 BCCP

A2 BCCP

B ACC

B1 BCCP

B2 BCCP

Figure : Rendering of ACC and ASCCP

### View CCs

The user should be able to search for a CC across all types of CCs and open it up for viewing.

The search shall look through both the DEN and Definition columns to return relevant results.

The user should be able to filter out some specific types of CCs.

In the case that an ASCC or BCC is selected for viewing, the user should be able to easily navigate to the owner ACC. From an ACC, relevant ASCCPs should be shown or at least the user should be able to query for them.

In addition to a typical CC specification data fields, current owner, state, last upated by user, and last updated timestamp should also be displayed.

The system may show the detail/properties of an ACC or ASCCP first rather than the whole tree. Only when the user has chosen to open the ACC or ASCCP then the whole tree is shown.

If the

### Create a new ACC (and an ASCCP)

To create a new ACC, the system may first create a blank ASCCP and ACC on the UI. The system forces an ASCCP to be created at the same time. So initially the UI would show a tree with a single node on the left pane of the window with a default ASCCP’s DEN such as “An ASCCP Property Term. An ACC Object Class Term”. The user can modify properties of the ASCCP and ACC on the right pane of the window.

See section 3.10.3 for how the ASCCP and ACC should be rendered. ACC and ASCCP current and history records have to be properly created.

In addition to editing the ASCCP and ACC properties, the user must be able to invoke the following functions through the UI:

1. Make the ACC based on another ACC. This may be offered to the user by right clicking on the ACC in the background.
2. Add a child association to the ACC. This may be offered to the user by right clicking on the ACC (in the background).
3. Delete a child association from the ACC.
4. Insert a child association to the ACC.
5. Edit/Revise an association.
6. Create a new ASCCP.
7. Create a new revision of an ASCCP. After adding an association to an existing ASCCP that is in published state. The user may wish to modify it.
8. Create a new BCCP.
9. Create a new revision of a BCCP. After adding an association to an existing BCCP that is in published state. The user may wish to modify it.

See other subsections for details of these functionalities.

#### ACC current record

Populate the ACC table as follows.

|  |  |  |
| --- | --- | --- |
| ACC\_ID = Auto-generate database key. | | |
| ACC\_GUID = Create a new GUID. | | |
| Object\_Class\_Term = Give a default object class term as “A new ACC Object”. The user can modify this. | | |
| DEN = Object\_Class\_Term + “. Details”. | | |
| Definition = Blank. The user can modify this later. | | |
| Based\_ACC\_ID = Null. | | |
| Object\_Class\_Qualifier = Null. | | |
| OAGIS\_Component\_Type = Default to 1, Semantics, but the user can change to 0 – 4 (show the label on the UI not the int value). | | |
| Module = If this is a user created CC (not a model CC), leave this null. However, label it “Inherited from the Extension component” and make it uneditable. Otherwise (it is a model CC), leave this blank initially and the user has to specify a value. | | |
| Namespace\_ID = Leave this null. However, if it is a user created CC, label it “Inherited from the Extension component” and make it uneditable. Otherwise, label it “Assigned at the release level” and make it uneditable. | | |
| Created\_By\_User\_ID = The logged in user. | These are referred to was WHO columns. | These are referred to as History columns. |
| Owner\_User\_D = The logged in user. |
| Last\_Updated\_By\_User\_ID = The logged in user. |
| Creation\_Timestamp = Current time. |
| Last\_Update\_Timestamp = Same as Creation\_Timestamp. |
| State = 1. **Note:** 1 means Editing. |  |
| Revision\_num = 0. |  |
| revision\_trackiing\_num = 0. |  |
| Revision\_Action = Null. |  |
| release\_id = Null. |  |
| Current\_ACC\_ID = Null. |  |

#### ACC history record

Follow general guideline.

#### ASCCP current record

The data should represent a default ASCCP for the above ACC as discussed at the beginning of 3.10.5. Its Module and Namespace\_ID column shall follow the same logic as that of the ACC current record above. The Reusable\_Indicator column shall be first set to true.

#### ASCCP history record

Follow general guideline.

### Create a new revision of an ACC

Pre-condition: ACC must be in the Published state. The log-in user has to be the owner of the ACC.

Get all the child associations including the ASCCs and BCCs in the current record. They need to be checked out along with their parent ACC. Their history columns must be properly updated and the corresponding history records get created before showing the UI for the user to be able to edit the ACC.

On this UI, the user shall be able to do invoke all the functions listed in 3.10.5, except making/changing the ACC to based on another ACC.

#### Notification

Need to design this in the Architecture section first.

### Edit an ACC

### Add a child association to an ACC

When this use case is invoked, the invoking use case must pass the parent/owner ACC of the association to be created. We’ll refer to the parent ACC as pACC.

A UI must be brought up for the user to be able to search for an ASCCP and BCCP. The search must look through only those (current) records. The entities in different states should be shown differently. Entity in Deprecated state must not be shown for reuse. In addition, those records with Reusable\_Indicator = False must be filtered out. The search must look through the property term and definition of the ASCCP and BCCP as well as the object class term and definition of the ACC, to which the ASCCP points. The user must be able to select an ASCCP or a BCCP, from which an ASCC or BCC record shall be created, respectively. Let tProperty be the target BCCP or ASCCP the user has selected.

The sequencing key of the new association, seqKey, must be either passed from the invoking use case or input by the user after the ASCCP or BCCP . For the case of BCCP, the UI needs to ask the user “Do you want to insert the BCCP as a meta-data (attribute)?” This will indicates the entityType to be used to populate the BCC record (recall that the value is 0 for attribute and 1 otherwise). Also, the the entity type is attribute then the seqKey is assigned the value 0.

#### Populate the ASCC current record

|  |
| --- |
| ASCC\_ID = Auto-generate database key. |
| ASCC\_GUID = Generate a new one. |
| Cardinality\_Min = initialize with 0. |
| Cardinality\_Max = initialize with 1. |
| Sequencing\_Key = seqKey. |
| Assoc\_From\_ACC\_ID = pACC. ACC\_ID. |
| Assoc\_To\_ASCCP\_ID = ASCCP\_ID of tProperty. |
| DEN = pACC.Object\_Class\_Term + “. “ + tProperty.DEN |
| Definition = Null. The user can edit this later on. |
| Created\_By\_User\_ID = The logged in user. |
| Owner\_User\_ID = The logged in user. |
| Last\_Updated\_By\_User\_ID = The logged in user. |
| Creation\_Timestamp = Current time. |
| Last\_Update\_Timestamp = Same as Creation\_Timestamp. |
| State = 1. **Note:** 1 means editing. |
| revision\_num = 0. |
| revision\_trackiing\_num = 0. |
| Revision\_Action = Null. |
| release\_id = Null. |
| Current\_ASCC\_ID = Null. |

#### Popultate the ASCC history record

The data should be the same as that in the ASCC current record except below columns.

|  |
| --- |
| ASCC\_ID = Auto-generate database key. |
| revision\_num = 1. |
| revision\_trackiing\_num = 1. |
| Revision\_Action = 1. |
| release\_id = Null. |
| Current\_ASCC\_ID = ASCC\_ID of the ASCC current record. |

#### Populate the BCC current record

A BCC current record is populated in the same way as the ASCC record. The only different is that the Entity\_Type column needs to be populated, additionally. This takes the value from the user input – entityType – described earlier.

#### Populate the BCC history record

The BCC history record is populated in the same way as the ASCC history record with the exception that the Current\_BCC\_ID column must point to the BCC\_ID of the above BCC current record.

### Insert a child association to an ACC

Similar to the case of Add a child association to an ACC use case in 3.10.8, the invoking use case must pass the pACC and seqKey and allow the user to select a tProperty. The user must also indicates whether the BCCP is to be inserted as a meta-data/attribute, in which case the entityType variable is assigned 0 otherwise it is assigned 1. The corresponding ASCC and BCC current and history records must be created as in 3.10.8.

In addition, in the case that the entityType is not 0, the associations having the same sequencing key as the seqKey and higher must have their sequencing keys shifted up by one. That is, the ASCC and BCC currernt records must have their Sequencing\_Key column updated; and the corresponding new ASCC and BCC history records created. The way to populate the history record is the same as in 3.10.8. For the history record, only the revision\_trackiing\_num needs to be incremented and the Revision\_Action shall be 2 (update).

It should be noted that the Add a child association to an ACC use case in 3.10.8 may be considered a specific case of this use case where there is no need to do the sequecing key shifting of other associations and they both may be implemented as a single overloaded function.

### Delete a child association from an ACC

The invoking use case passes the association, dAssoc, to be deleted. This can be an ASCC or a BCC.

The current record corresponding to the dAssoc has to be deleted from the ASCC or BCC table. A history record has to be created.

The history record has to be created similar to that in 3.10.8 for ASCC or BCC deletion. That is all data are essentially the same as that of the current record. However, the revision\_trackiing\_num has to be incremented by one from the latest history record. The Revision\_Action has to be 3 (delete). In addition, the Last\_Update\_Timestamp needs to be replaced with the current time.

The sequencing key of the associations after the deleted association has to be updated if the deleted association is not a BCC whose Entity\_Type is 0. The sequencing key of the current records have to be reduced by 1 and then their history records shall be be properly created.

### Edit/Revise an association

The user can change cardinality and definition of an ASCC or a BCC. In addition, the user can change the BCC’s entity type. For these cases, the current record of the association is simply updated with the new value or values and the Last\_Update\_Timestamp is updated to the current time. A corresponding new history record needs to also be created with revision\_trackiing\_num incremented and Revision\_Action = 2 (Update).

The user may also change the position (sequencing key) of the association. This is better provided to the user by dragging the association up or down. Since changing a position involves updating several association records, it will be a good idea to ask for the user confirmation first. Alternatively, the position change maybe provided in a way that encourages several changes in a single transaction. For example, the user specifically selects a menu to revise association positions and the system then locks the user to be able to make only that particular changes. Then, the user click on a menu to end the transaction and the system figures out which association positions were changed and correspondingly updated them.

### Make an ACC based on another ACC

### Create a new ASCCP

### Create a new revision of an ASCCP

### Create a new BCCP

### Create a new revision of a BCCP

### Deprecation

### Undoing

### Event notification

## Generate GUID

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

Regarding BDT for code list. This is an idea.

# Configuration File

## SQL for Top-level Concept

<topLevelAsccpSql>This must be a valid condition for SQL statement based on the database schema to get a list of top-level ACC.</topLevelAsccpSql>

Example:

<topLevelAsccpSql>select \* from ASCCP where Module != “Components” and Module != “Meta” and Module != “Noun” and Module != ‘Extensions’.</topLevelAsccpSql>

## Expression type mapping display configuration

<expressionTypeMapping isDefault=”True”> <!-- isDefault indicates that this expression is a default. Only one expressionTypeMapping should have this value as True. If there are multiple, the first one that is true will be used. -->

<name>XSD Built-in Type</name><!-- pretty-print name of this configuration used in the view object -->

<builtInTypeTable><!-- primitive data type table of this expression -->

<name>XSD\_BuiltIn\_Type</name><!-- name of the table -->

<primaryKeyColumn>XSD\_BuiltIn\_Type\_ID</primaryKeyColumn><!-- primary key of the table -->

<foreignKeyColumnInMappingTable>XSD\_BuiltIn\_Type\_ID</foreignKeyColumnInMappingTable><!-- Foreign keys in the CDT\_Allowed\_Primitive\_Expression\_Type\_Map and the CDT\_SC\_Allowed\_Primitive\_Expression\_Type\_Map tables which point to the above primary key. -->

</builtInTypeTable>

</expressionTypeMapping>

Another example

<expressionTypeMapping>

<name>Java Type</name>

<builtInTypeTable>

<name>Java\_BuiltIn\_Type</name>

<primaryKeyColumn>Java\_Type\_ID</primaryKeyColumn>

<foreignKeyColumnInMappingTable>Mapped\_Java\_Type\_ID</foreignKeyColumnInMappingTable>

</builtInTypeTable>

</expressionTypeMapping>

## UI related variables

<UILabels>

ABIEs =Profile BOD

ABIEs =Profile BOD2

<CreateTopLevelABIE>Create a Profile BOD</CreateTopLevelABIE>

# 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

1. Note here the term user CC vs. developer CC. User CCs are core components created by the OAGIS end users as part of the OAGIS extension. Developer CCs are components created and released by OAG as part of the OAGIS standard. [↑](#footnote-ref-1)