Corporate Services for the Natural Resource Sector

Information Management Branch

NRS Logical Data Model to Physical Data Model Transformations

Last Updated:	Dec 10th, 2016
Version:	2.1
Document:	NRS Logical Model to Physical Model Transformations.docx

Table of Contents

1. Version Control

Document Version	Date	Author(s)	Change Reference
1.0.0	Apr 1, 2015	Vivid Solutions Inc.	Initial Draft
1.1.0	May 7, 2015	Vivid Solutions Inc.	Updated to match physical model standards.
1.2.0	May 22, 2015	Vivid Solutions Inc.	Added support for enumerations. Minor fixes to scripts.
1.3.0	Aug 7, 2015	IMB Data Architecture, IMB DBA, NRPP DBA, Vivid Solutions	Clarifications and added custom NRS Logical to Physical model transformation XMI import reference data.
2.0.0	June 22, 2016	IMB Data Architecture	Changes: • added support for List of Values as a standard code / list validation type • added support for business keys and unique keys
2.1.0	Nov 10, 2016	IMB Data Architecture	Changes: • changed default primary key type from ID to GUID • changed association stereotype from identifying to Business Key

1. Purpose

This document describes the process of taking a logical data model in enterprise architect, transforming it to a physical database model, and finally generating the DDL scripts to create the database objects for the target database.

The transformation process uses custom logical to physical transformation templates created for the NRS to comply with the organizations database standards.

The Natural Resource Sector standards for Logical Modeling must be applied when creating a Logical Data Model. This document supplements those standards with instructions for successfully creating the physical data model.

Physical data model and DDL generation is a continuous improvement process with the goal of streamlining the process as much as possible. As such, logical to physical transformation scripts will be periodically updated and DDL transformation scripts may be created in the future.

1.2 Version Alignment

Although the current version of the logical data model to a physical data model transformation scripts goes a long way towards creating a NRS standard physical data model that can be implemented in NRS Oracle databases, these scripts will be updated to improve what is achieved in the physical model generation.

As the transformation scripts are expanded for improved functionality, improved versions will be available. These scripts and this document will be aligned through identifying version numbers.

This document aligns with the following.

Name	Version	Date	Type
NRS Data Modeling Standards with EA	3.1.0	Dec 2016	PDF doc
NRS Logical To Physical Transformation	2.1.0	Dec 2016	EA XMI import file
NRS Physical Data Modeling Standards	2.5.0	Dec 2015	PDF doc
BC Parks Reservations	TBD		EAP file
NRS Standards for Modeling with EA	4.1.0	Dec 2016	PDF doc

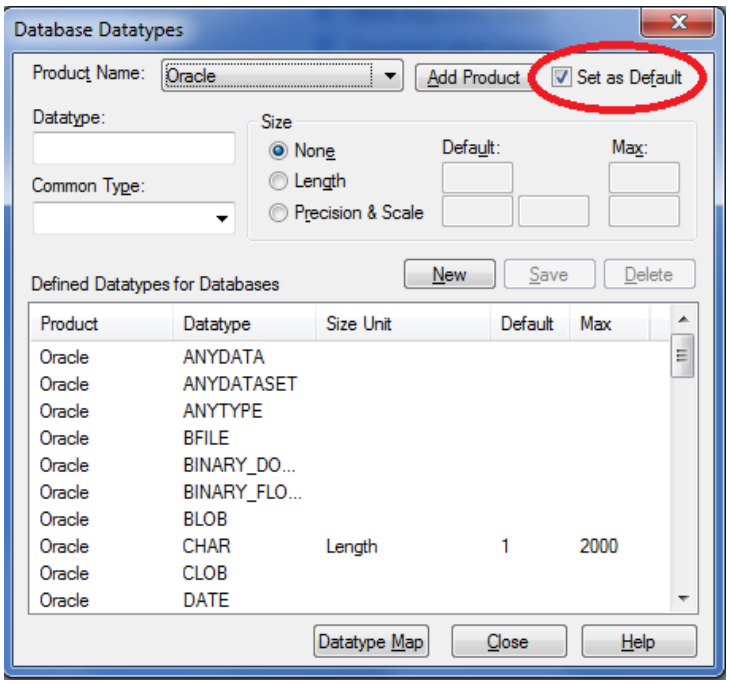
2. Enterprise Architect Configuration

Enterprise Architect must be configured to transform logical models into a physical data model using custom scripts that comply with the NRS modeling standards.

This section describes the configuration steps that must be followed to produce a successful transformation.

2.1 Set the default DBMS to "Oracle"

- EA 10 - Settings->Database Datatypes menu option.
- EA 12 - Project → Settings->Database Datatypes menu option.
- In the Product Name drop-down list, select "Oracle"
- Check the *Set as Default* check box:



2.2 Configure Stereotypes

Stereo types need to be created to mark attributes and associations as Business Key. Attributes and associations marked as Business Key indicate the corresponding columns will transformed to be part of a unique business key.

To configure the new attribute stereotypes:

1. Navigate to the menu item **Setting->UML Types…**
2. Click the **New** button.
3. Set the **Stereotype:** field to the value "Business Key".
4. The **Group name:** field should remain blank.

5. Set the **Base Class:** field to the value “attribute” in the drop down list.
6. Set the **Notes:** field to the value “Business Key”
7. Click the **Save** button to save the stereotype.

To configure the new association stereotypes:

1. Navigate to the menu item **Setting->UML Types…**
2. Click the **New** button.
3. Set the **Stereotype:** field to the value “Business Key”.
4. The **Group name:** field should remain blank.
5. Set the **Base Class:** field to the value “association” in the drop down list.
6. Set the **Notes:** field to the value “Business Key”.
7. Click the **Save** button to save the stereotype.

The end result is that there should be 2 stereotypes created called “Business Key”, one is of type “attribute” and the other is of type “association”.

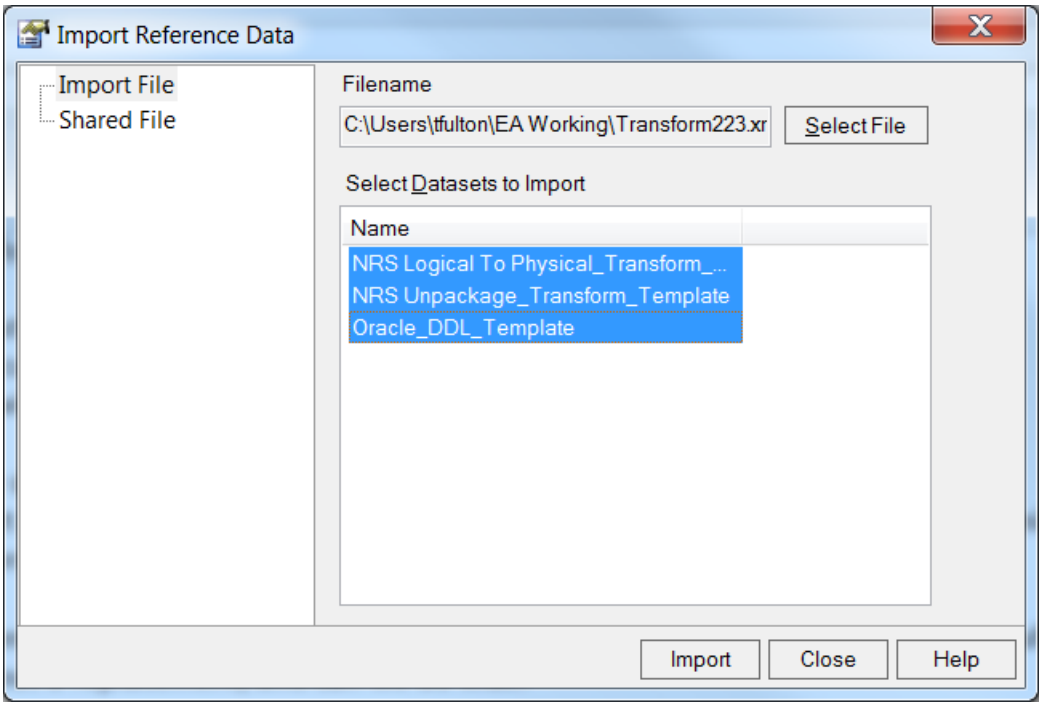
2.3 NRS DDL Transformation Templates

A custom NRS Logical to Physical XMI reference data import file has been created to import custom transformation templates which support transformation of logical data models to physical database models targeted for implementation in an Oracle database.

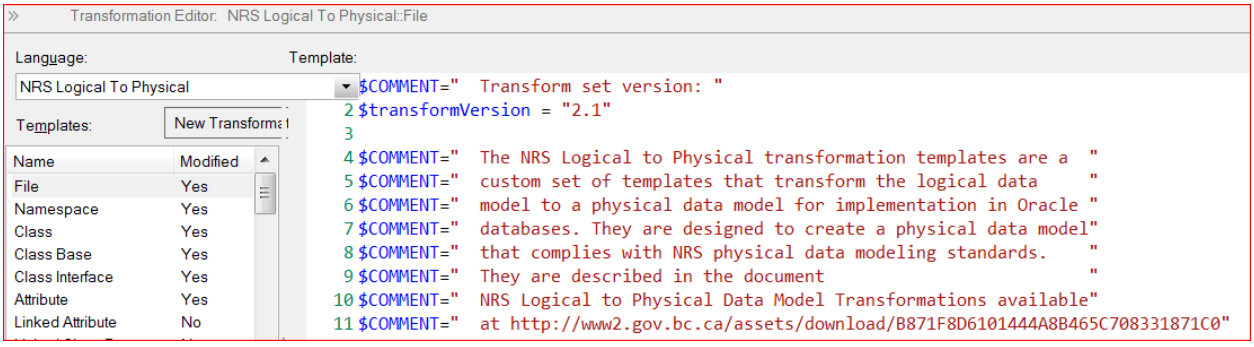
Note: recent versions of the transformation have added functionality which has caused the transformation execution to be slow in completing larger models. If your experience slow transformation performance (>10 minutes) please contact the Data Architect assigned to your project or CSNR.Data.Architecture.Services@gov.bc.ca

To import the template

- download the NRS Logical to Physical XMI file saving it with a XML file type
- EA 10 - Project → Model Import/Export → Import Reference Data
- EA 12 - Project → Data Management → Import Reference Data
-
- browse for and select the NRS Logical to Physical XMI reference data file
- select the NRS Logical To Physical_Transform_Template and Import
- the Oracle DDL Template can also be imported to create sequences to NRS naming standards
- NRS Unpackage Transform Template is a work in progress



After import the NRS Logical To Physical transformation scripts will look like this:



To run the transformation, see section 4 – NRS Logical Model to Physical Model Transformation after completing the logical model preparation in section 3.

3. Logical Data Model Preparation

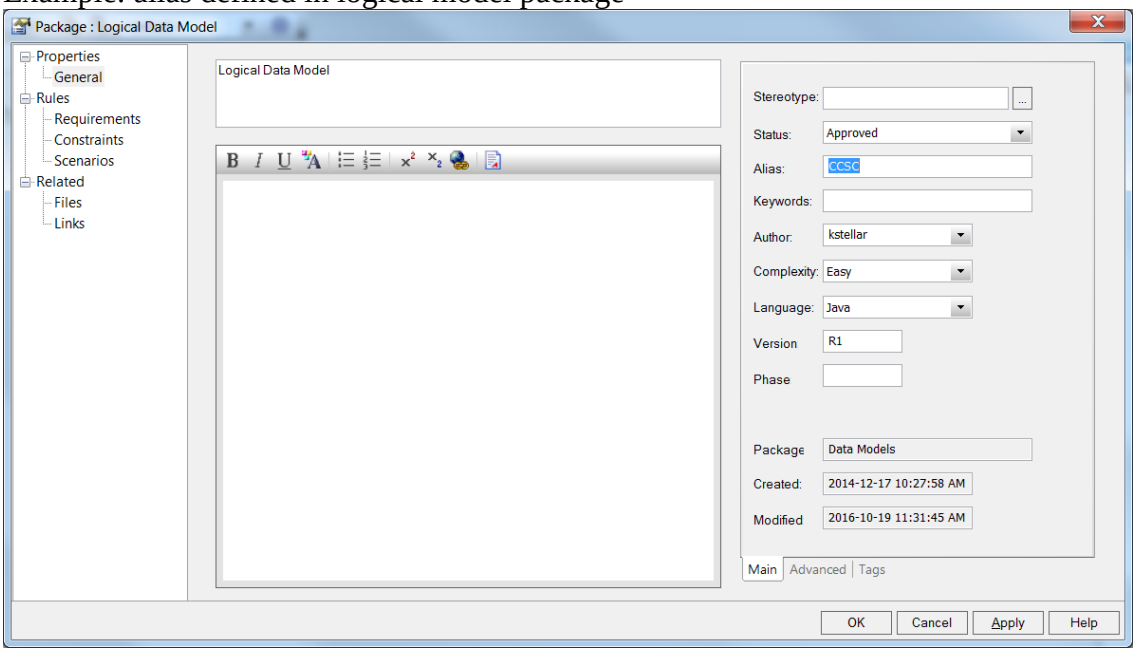
This section describes properties that determine how the logical data model is a transformed to a physical data model. The transformation process relies on a set of mandatory properties that are required and must be set in the logical data model at the class and attribute level. The section describes the properties that must be configured and the properties that are optional.

3.1 Package Configuration

3.1.1 Package Alias

Name	Alias
Type	Package Property
Required	Yes
Default	If an Alias is not specified, then the alias will default to ‘SET_PACKAGE_ALIAS_TO_APPLICATION_ACRONYM’.
Purpose	The alias is the application Short Name as defined in the Infrastructure Reporting System (IRS). It is assigned to the logical model package and used in the logical to physical model transformation to assign the tablespace name.
Constraints	1. Alias must be short name for the application or service from IRS.

Example: alias defined in logical model package

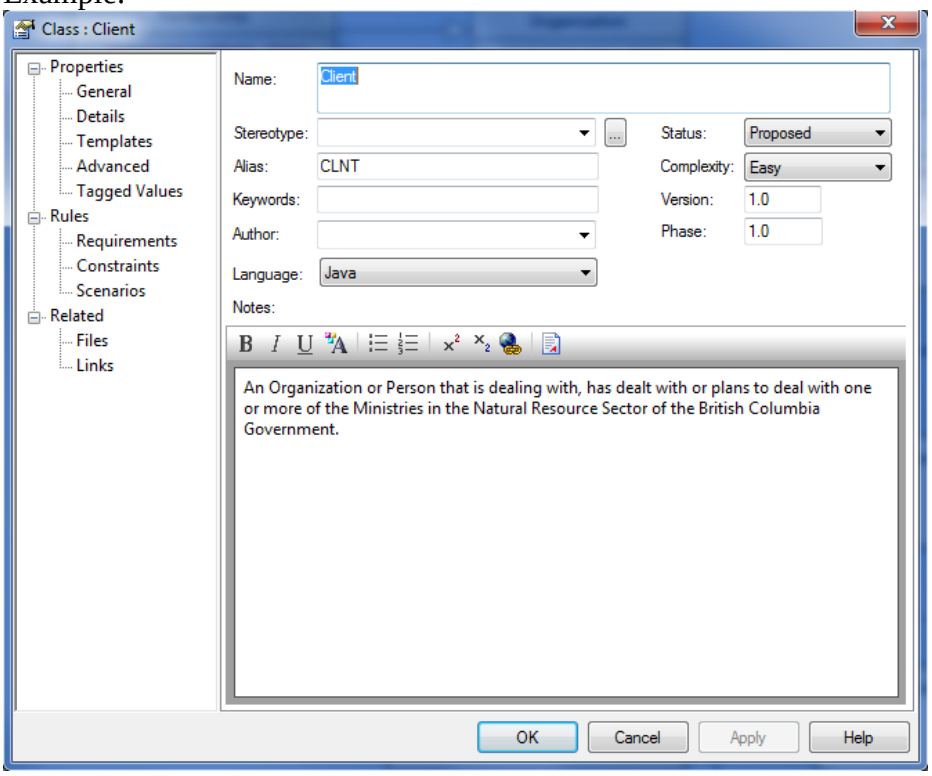


3.2 Class Configuration

3.2.1 Table Name

Name	Name
Type	Class Property
Required	Yes
Default	None
Purpose	The logical name of the class that is to be transformed. If a primary key is generated automatically from the class name, then the primary key name will be derived from the class name and will be suffixed with the primary key type. The length of class names should take into consideration what the resulting primary key column will look like with the primary key suffix. For example, if a GUID primary key is to be generated for a "Client" table, then the primary key will be named "CLIENT_GUID" If an ID primary key is to be generated for a "Client" table, then the primary key will be named "CLIENT_ID"
Constraints	1. The first character of each word in the name must be capitalized. 2. Words in the name are separated by the space character. 3. Take into account the type of primary key created for the table as name length limitations include primary key suffix.

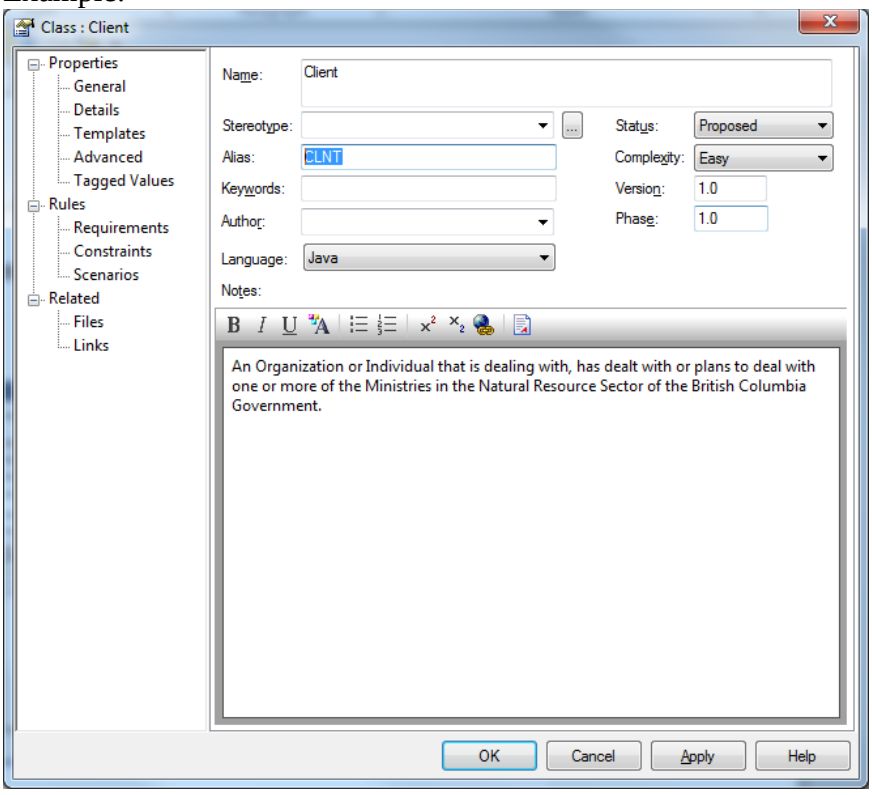
Example:



3.2.2 Table Alias

Name	Alias
Type	Class Property
Required	Yes
Default	If an Alias is not specified, then the alias will default to the first 8 characters of the class name (this runs the risk of the alias being non unique).
Purpose	The alias is a short name for the table and is used to construct foreign key names, in the physical model. Names can be created using the first letter of each word that makes of the table name. For example a table with the name "Inspector Assigned Establishment" could have a short name of "IAE". Approved or generally recognized abbreviations can also be used for the short name. For example, a table with the name "Client" could have a short name of "CLNT". The idea is to be able to recognize the full table name by the short name.
Constraints	2. Alias must be 8 characters or less long. 3. Alias may be alpha-numeric 4. Alias must be unique within the database schema

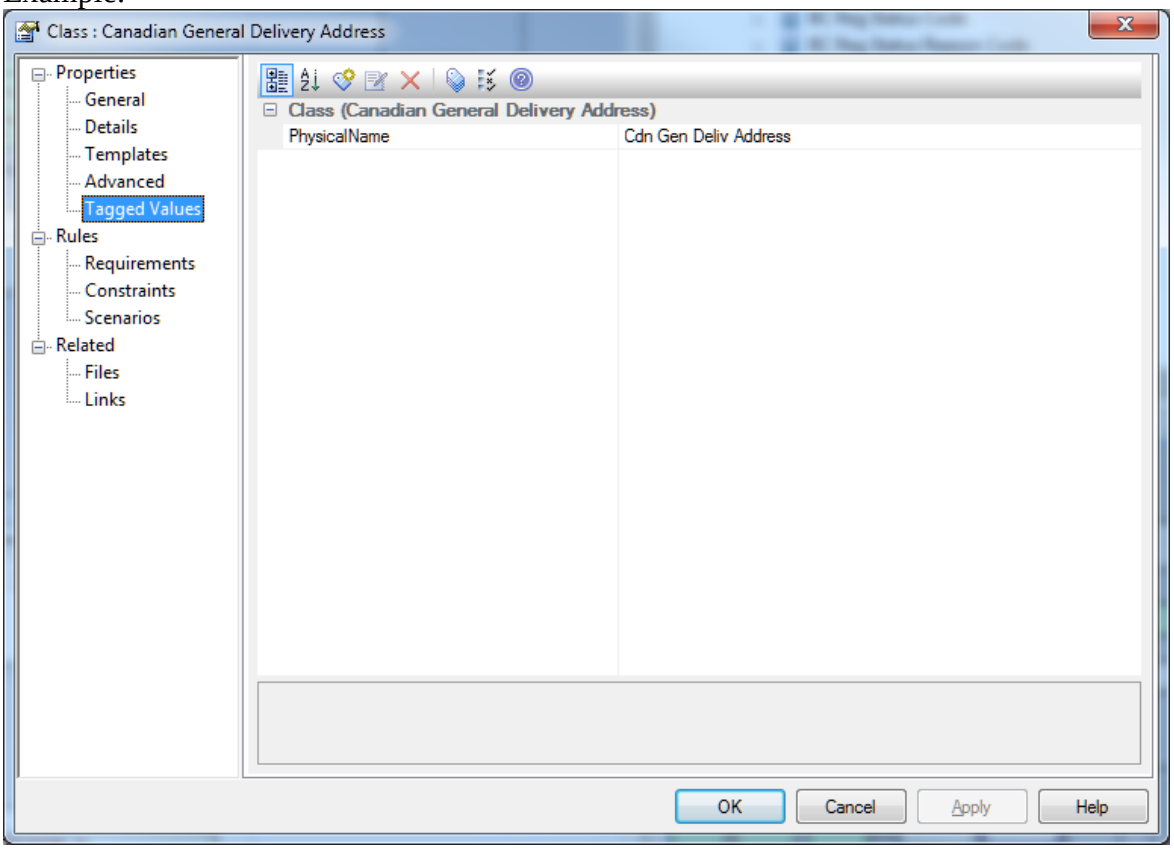
Example:



3.2.3 Table Physical Name

Name	PhysicalName
Type	Tagged Value
Required	No
Default	If a Physical Name is not specified then the first 30 characters of the class name will be used.
Purpose	In a logical model a class name may exceed the allowable 30 character length for table names in an Oracle Database. For example: Canadian General Delivery Address A physical name tagged value must be specified for the class that specifies a 30 characters or less physical name to use for the table in the physical model.
Constraints	1. Physical Name must be 30 characters or less long. 2. The first character of each word in the name must be capitalized. 3. Words in the name are separated by the space character. 4. Take into account the type of primary key created for the table as name length limitations include primary key suffix.

Example:



3.2.4 Table Primary Key

For ISSS new development all new surrogate keys must be GUIDs so where a surrogate key is being used as the table primary key the PrimaryKeyType will need to be GUID or not identified in which case it will default to GUID.

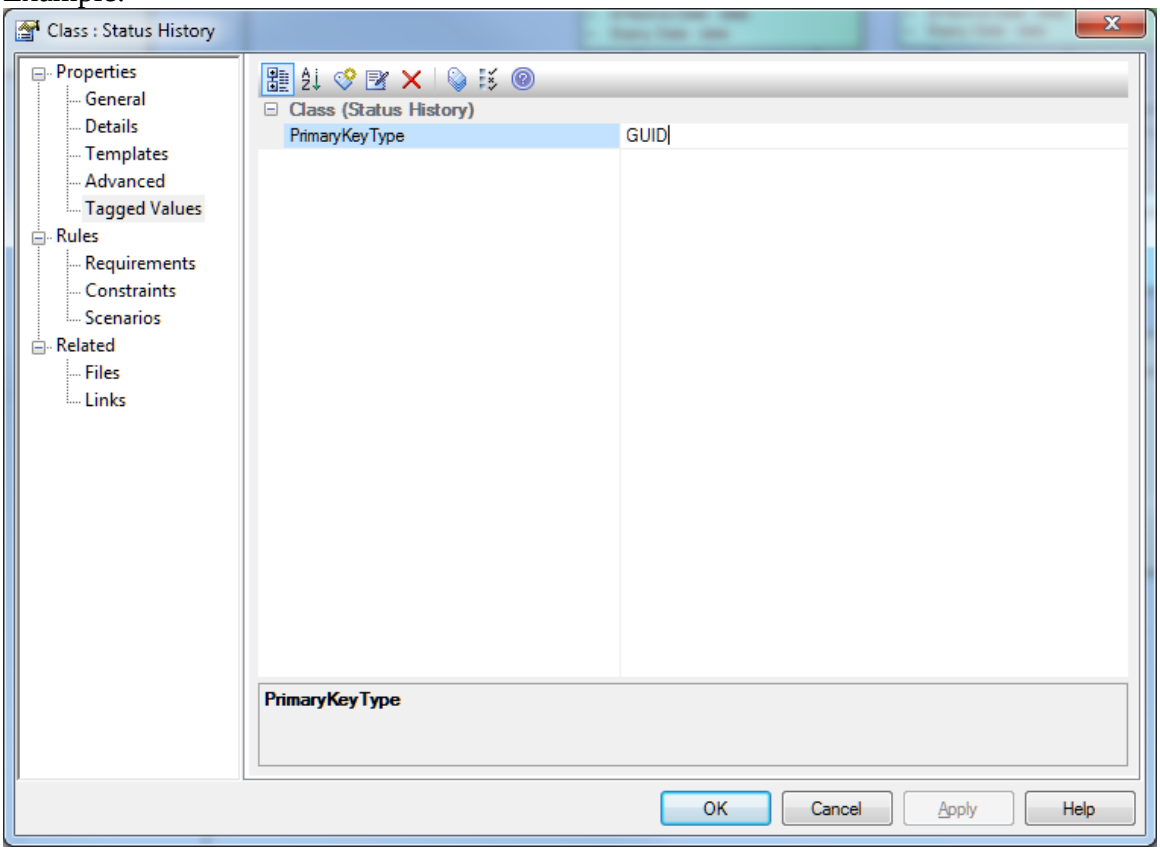
Principles for GUID use

- provide unique record identifiers (values) both within and across databases
  - can create new key values without having to worry that the value might already be used
  - can be generated outside the database and be unique without blocking
- provide a standard key format which supports consistent use
  - within web services
  - for associating data from multiple sources to common data such as client data or administrative boundary data
- must be in standard UUID format

As use of GUIDs is evolving, discuss with the Data Architect assigned to your project if use is a concern for your project.

Name	PrimaryKeyType
Type	Tagged Value
Required	Yes
Default	If a PrimaryKeyType tag value is not specified then it will default to the type "GUID"
Purpose	The PrimaryKeyType tag is used to specify how the primary key for the table is to be created. A primary key can be automatically generated for the table. An automatically generated primary key can be of one of the following types: • GUID – A surrogate key of type RAW(16) that stores a GUID value. • ID - A surrogate key of type Number(15) that stores a sequence value. • CODE – A key of type VARCHAR2(10) that stores a unique code value. • LIST – A key of type VARCHAR2(25) that stores a unique list value. • Business Key – A key of business column(s) that store business data values.
Constraints	1. Must be set to one of the following (GUID, ID, CODE, LIST, Business Key) or a GUID primary key will be generated. 2. Where an existing application has used ID surrogate keys a GUID might not be a practical choice. Discuss with the project DA if new ID surrogate keys are planned.
Warning	

Example:



3.2.4.1 GUID Primary Key

If the "PrimaryKeyType" class tag is configured with the value "GUID", or no PrimaryKeyType class tag is added, then a primary key column will be created for the table with the following characteristics:

- The primary key name will be derived from the table name or the physical name defined for the table.
- The primary key name will have the suffix "_GUID".
- The primary key field will have the data type RAW(16) which is suitable for storing a GUID value.
- The default value for the GUID column will be set to SYS_GUID().

Example:

- If the table is named "Some_Table_Name", then the primary key column will be named "SOME_TABLE_NAME_GUID".
- If the table is named "Some_Super_Long_Table_Name" and the table has a PhysicalName tag defined as "Some_Sup_Lng_Tbl_Name", then the primary key column will be named "SOME_SUP_LNG_TBL_NAME_GUID".

3.2.4.2 ID Primary Key

If the "PrimaryKeyType" class tag is configured with the value "ID", then a primary key column will be created for the table with the following characteristics:

- The primary key name will be derived from the table name or the physical name defined for the table.
- The primary key name will have the suffix "_ID".
- The primary key field will have the data type NUMBER(15) which is suitable for storing a sequence values.
- The AutoNum property on the ID column will be set to True with a StartNum of 1 and an Increment of 1 so a sequence will be created to populate the surrogate key during DDL generation.

Example:

- If the table is named "Some_Table_Name", then the primary key column will be named "SOME_TABLE_NAME_ID".
- If the table is named "Some_Super_Long_Table_Name" and the table has a PhysicalName tag defined as "Some_Sup_Lng_Tbl_Name", then the primary key column will be named "SOME_SUP_LNG_TBL_NAME_ID".

3.2.4.3 CODE Primary Key

If the "PrimaryKeyType" class tag is configured with the value "CODE", then a primary key column will be created for the table with the following characteristics:

- The primary key name will be derived from the table name or the physical name defined for the table.
- The primary key name will have the suffix "_CODE".
- The primary key field will have the data type VARCHAR2(10) which is suitable for storing a code value.

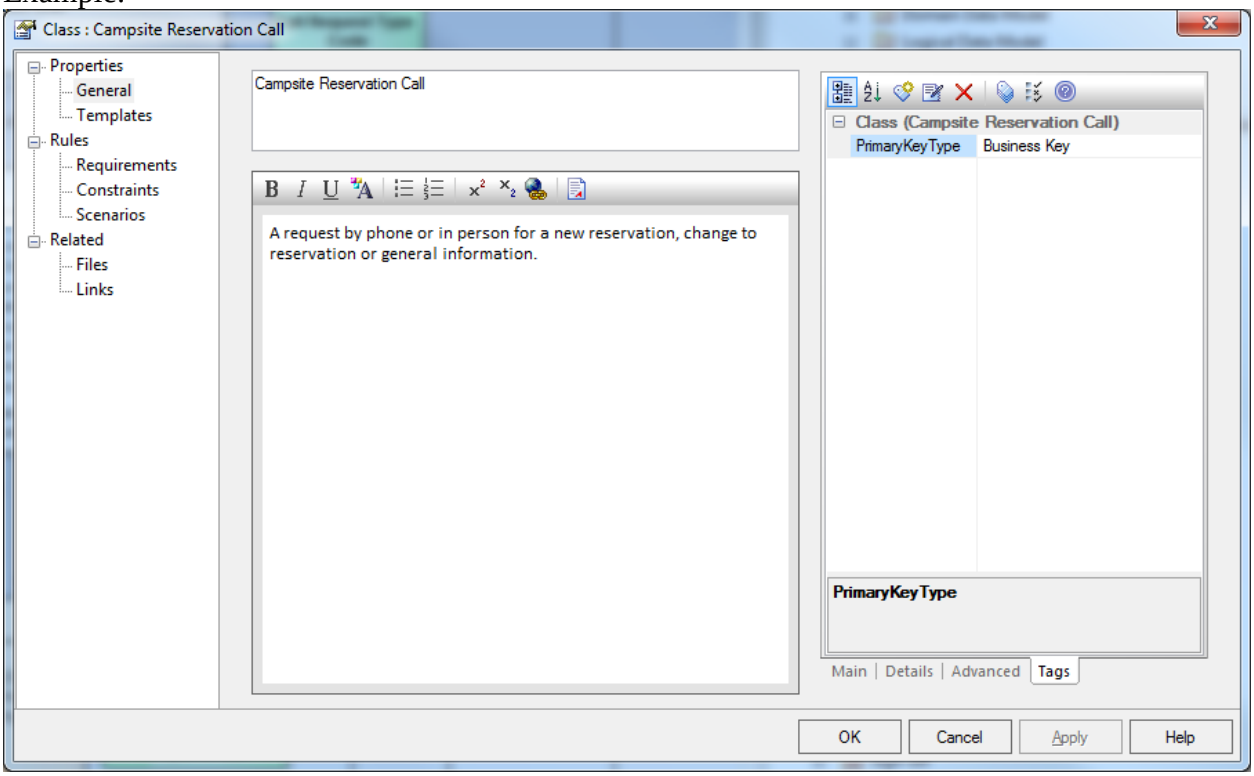
Example:

- If the table is named "Some_Table_Name", then the primary key column will be named "SOME_TABLE_NAME_CODE".
- If the table is named "Some_Table_Name_Code", then the primary key column will be named "SOME_TABLE_NAME_CODE".
- If the table is named "Some_Super_Long_Table_Name" and the table has a PhysicalName tag defined as "Some_Sup_Lng_Tbl_Name", then the primary key column will be named "SOME_SUP_LNG_TBL_NAME_CODE".

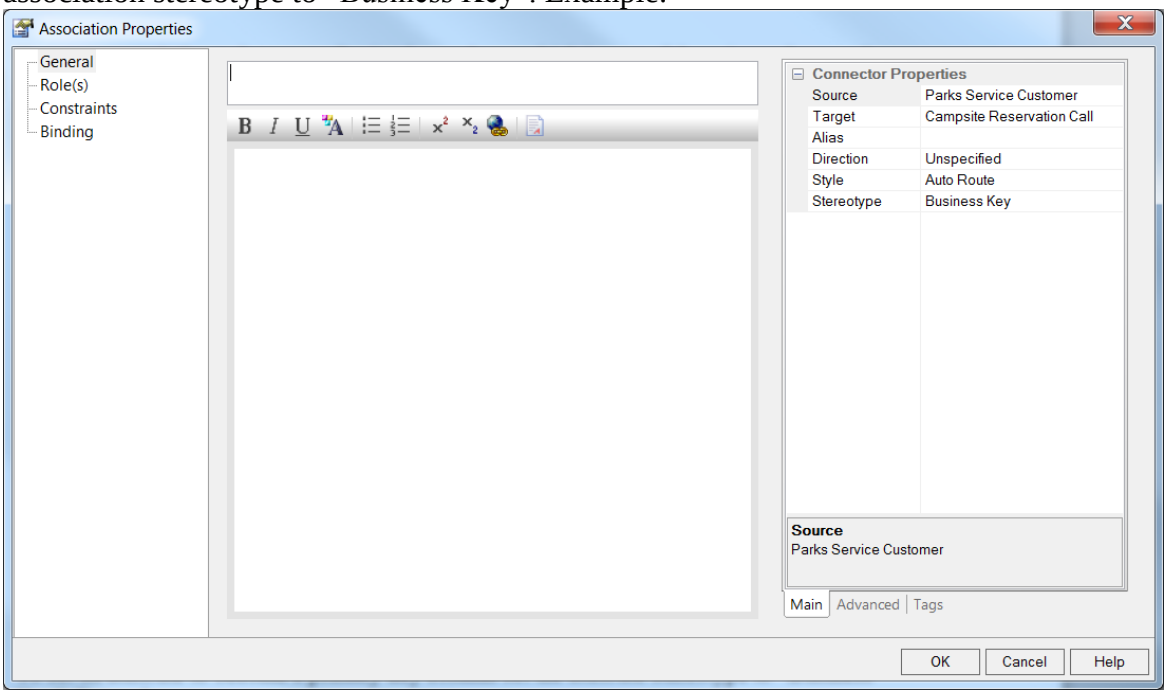
3.2.4.4 Business Key Primary key

If the "PrimaryKeyType" class tag is configured with the value "Business Key", then a primary key is created from any associations and any attributes with a stereotype of "Business Key". The transformation script will set one or more columns, or foreign key columns, as the primary key

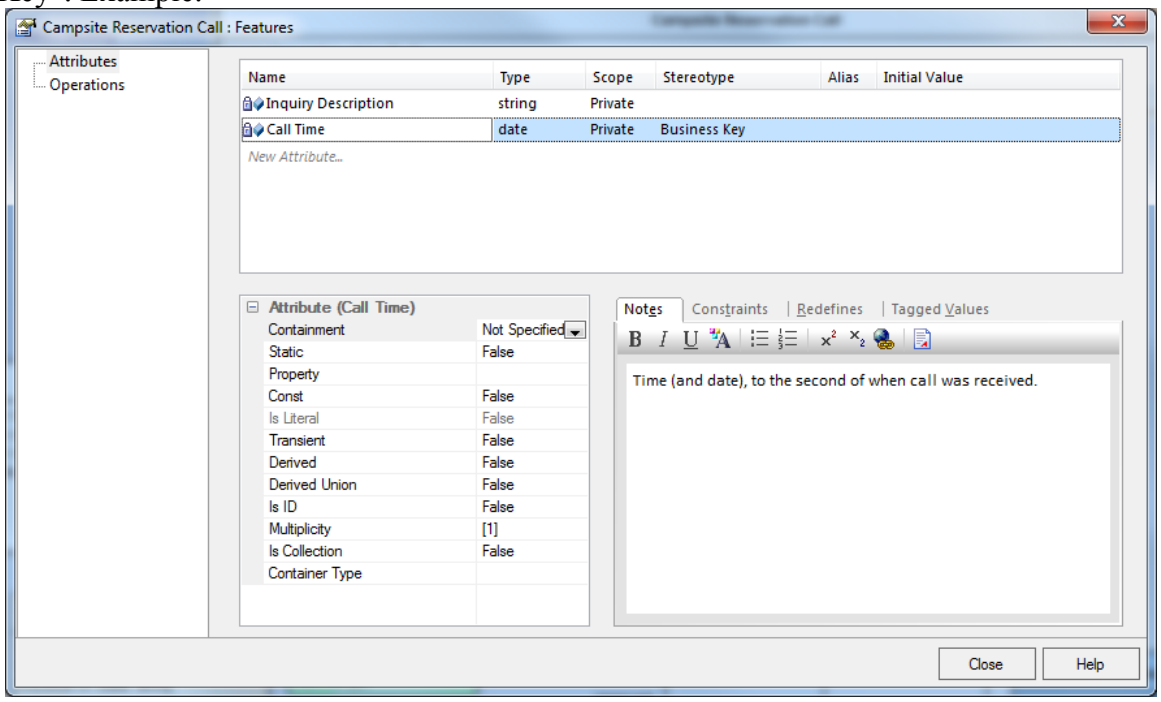
Example:



To set an association to become foreign key column(s) which is part of the primary key, set the association stereotype to “Business Key”. Example:



To set an attribute to become a primary key column set the attribute stereotype to “Business Key”. Example:



The transformation process will determine that the class is configured to generate primary keys from Business Key associations. The process will scan for associations to the class with a stereotype of Business Key and add foreign key columns created from the association to the primary key constraint.

The transformation process will determine that the class is configured to generate primary keys from Business Key attributes. The process will then scan through the attributes of the class searching for the attributes with a stereotype of Business Key and add the fields to the primary key constraint.

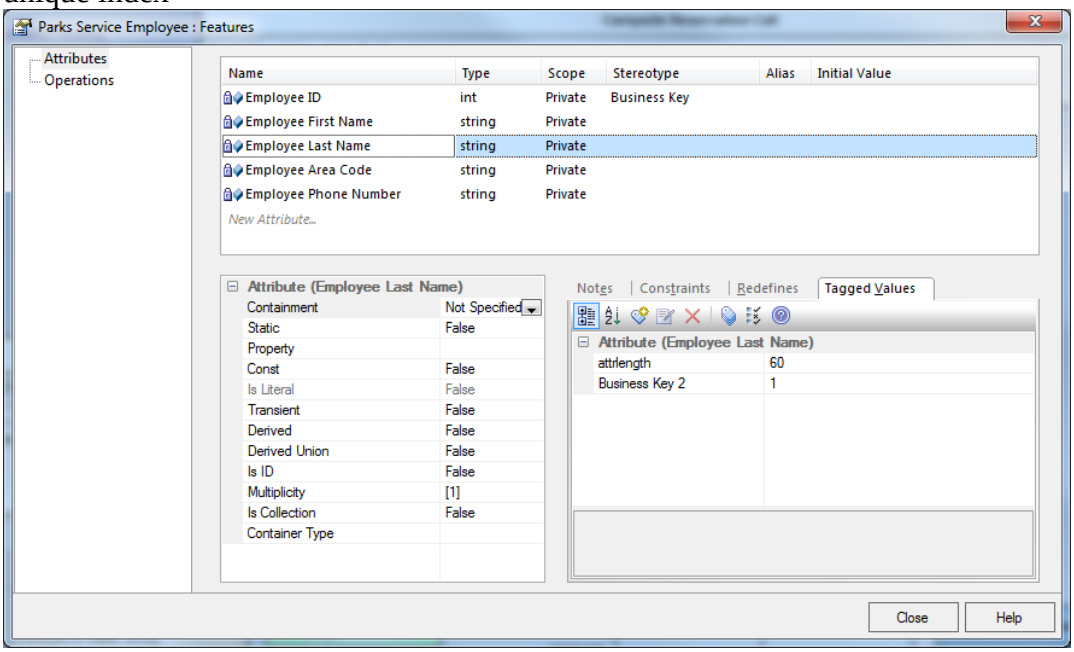
NOTE: Check to ensure primary key constraints from foreign key columns and business key columns are correct and columns are in the right order.

If the PrimaryKeyType is something other than Business Key (GUID, ID, CODE, LIST) and the above Business Key stereotypes are set then the transformation script will create a unique constraint rather than a primary key constraint.

3.2.5 Other Unique Keys

Name	Business Key 2, Business Key 3
Type	Tagged Value
Required	No
Default	If a Business Key 2 or Business Key 3 tag value is specified for the attribute then the attribute will become a unique constraint, or part of a unique constraint.
Purpose	The Business Key 2 or Business Key 3 tag is used to specify the attribute is a unique identifier or one attribute in a multi-attribute unique identifier. During physical model transformation any attribute with a Business Key 2 tagged value will become a column in a unique index with any other Business Key 2 attribute. During physical model transformation any attribute with a Business Key 3 tagged value will become a column in a unique index with any other Business Key 3 attribute. The class must also have a Business Key 2 or Business Key 3 tag specified. The tag value will become the unique index alias. E.g. Business Key 2 tag with value BK2COL for unique index alias name. Order of the columns in the unique index is determined by the value given to the Business Key 2 or Business Key 3 tagged values specified for the attributes.
Constraints	1. Must be set to one of the following (Business Key 2, Business Key 3)
Warning	Check to ensure the order of the columns is correct in the unique constraint added.

Example: tagged value to make attribute Employee Last Name first column in Business Key 2 unique index

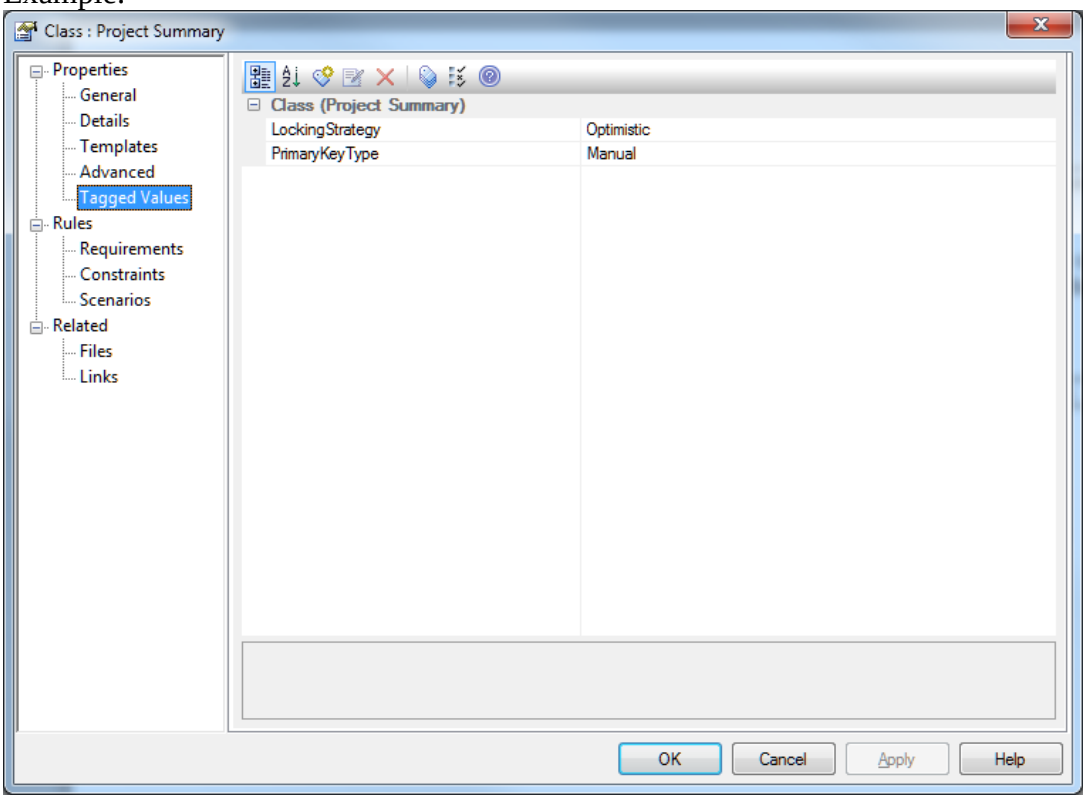


Support for additional business keys (4,5) has been added to the transformation as of version 2.1.0.

3.2.6 Table Locking

Name	LockingStrategy
Type	Tagged Value
Required	No
Default	If a "LockingStrategy" is not specified then the locking strategy will default to "Pessimistic"
Purpose	Locking refers to actions taken to prevent data in a relational database from changing between the time it is read and the time that it is used. An application locking strategy can be either *optimistic* or *pessimistic*. If an optimistic locking strategy is chosen then a column is added to the table to track revisions. The following optimistic locking column will be generated for the table: REVISION_COUNT (NUMBER(10))
Constraints	1. The "LockingStrategy" tag can have the values (Optimistic | Pessimistic).

Example:



3.3 List of Values

The NRS Logical To Physical data model transformation uses PrimaryKeyType tag of **List** to create list of values tables and a PrimaryKeyType tag of **Code** to create code tables.

Note: Enumerated objects in previous versions of the transformation were transformed to code tables. As of this version of the NRS_Logical_to_Physical_Transformation scripts it does not matter if it is an Enumerated Object or a Class being transformed, the type of transformation is determined by the PrimaryKeyType tagged value. An enumerated object will default to a List table unless a PrimaryKeyType tag of Code is added in which case a code table will be generated.

3.3.1 List Configuration

Classes or enumerated objects with a PrimaryKeyType tagged value of LIST will be transformed into the following list table structure:
Table Name: <Class(or enumerated object) Name with _LIST appended>
Columns :

<List_Table_ Name>	VARCHAR2 (25) NOT NULL,
Display_Order	Number (3),
Effective_Date	SYSDATE NOT NULL,
Expiry_Date	DATE DEFAULT 9999-12-31 NOT NULL,
Create_User	VARCHAR NOT NULL,
Create_Date	SYSDATE NOT NULL,
Update_User	VARCHAR NOT NULL,
Update_Date	SYSDATE NOT NULL

The list column length will default to VARCHAR2(25). If a different length is needed for values in the list add a KeyLength tag to the class with a value of the length needed.

If a description is required for the list of values, add a DescribedList tag to the class with a value of "Yes". The Description column will default to VARCHAR2(200). If the Description column needs to be a different length then the default 200 add a DescriptionLength tag to the enumeration with a value of the length needed.

3.3.2 Code Configuration

Classes or enumerated objects to be used for code tables and identified with a PrimaryKeyType tagged value on the class of CODE will be transformed into the following code table structure:
Table Name: <Class Name with _CODE appended>
Columns :

```
<Code_Table_ Name>      VARCHAR2 (10) NOT NULL,
Description             VARCHAR2 (200) NOT NULL,
Display_Order           Number (3),
Effective_Date          SYSDATE NOT NULL,
Expiry_Date             DATE DEFAULT 9999-12-31 NOT NULL,
Create_User             VARCHAR NOT NULL,
Create_Date             SYSDATE NOT NULL,
Update_User             VARCHAR NOT NULL,
Update_Date             SYSDATE NOT NULL
```

The code column length will default to VARCHAR2(10). If a different length is needed for code values add a KeyLength tag to the class with a value of the length needed.

The Description column length will default to VARCHAR2(200). If the Description column needs to be a different length then the default 200 add a DescriptionLength tag to the enumeration with a value of the length needed.

3.4 Attribute Configuration

3.4.1 Attribute Names

Attribute names must be spaced mixed case such as “Client Status Code”. The transformation process will create the column name in the physical model as “CLIENT_STATUS_CODE”.

3.4.2 Column Physical Name

Name	PhysicalName
Type	Tagged Value
Required	No
Default	If a Physical Name is not specified then the first 30 characters of the attribute name will be used.
Purpose	In a logical model an attribute name may exceed the allowable 30 character length for column names in an Oracle Database. If the attribute name is over 30 characters a physical name tagged value must be specified for the attribute that specifies a 30 characters or less physical name to use for the column in the physical model.
Constraints	1. Physical Name must be 30 characters or less long. 2. The first character of each word in the name must be capitalized. 3. Words in the name are separated by the space character. 4. The 30 character length includes any applicable suffixes added

3.4.3 Audit Columns

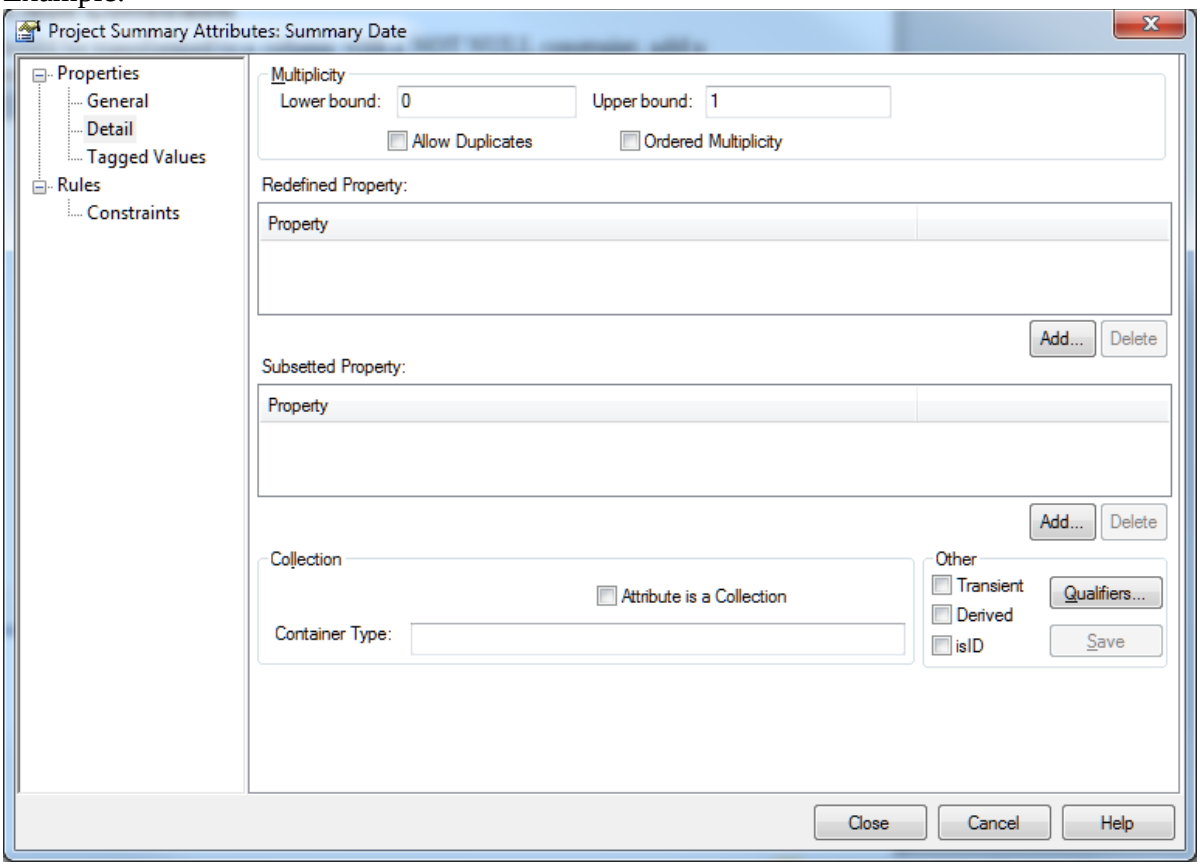
Purpose	Audit Columns will be automatically generated for each table. This ensures that audit columns are consistently named using ministry standards for all projects. The following audit columns will be generated for each table:
Default	The following audit columns will be generated for each table: • CREATE_USER (VARCHAR2(32) Not Null) • CREATE_DATE (DATE Not Null) • UPDATE_USER (VARCHAR2(32) Not Null) • UPDATE_DATE (DATE Not Null)

3.4.4 Attribute Not Null Constraint

Name	Lower bound
Type	Attribute Property
Required	No

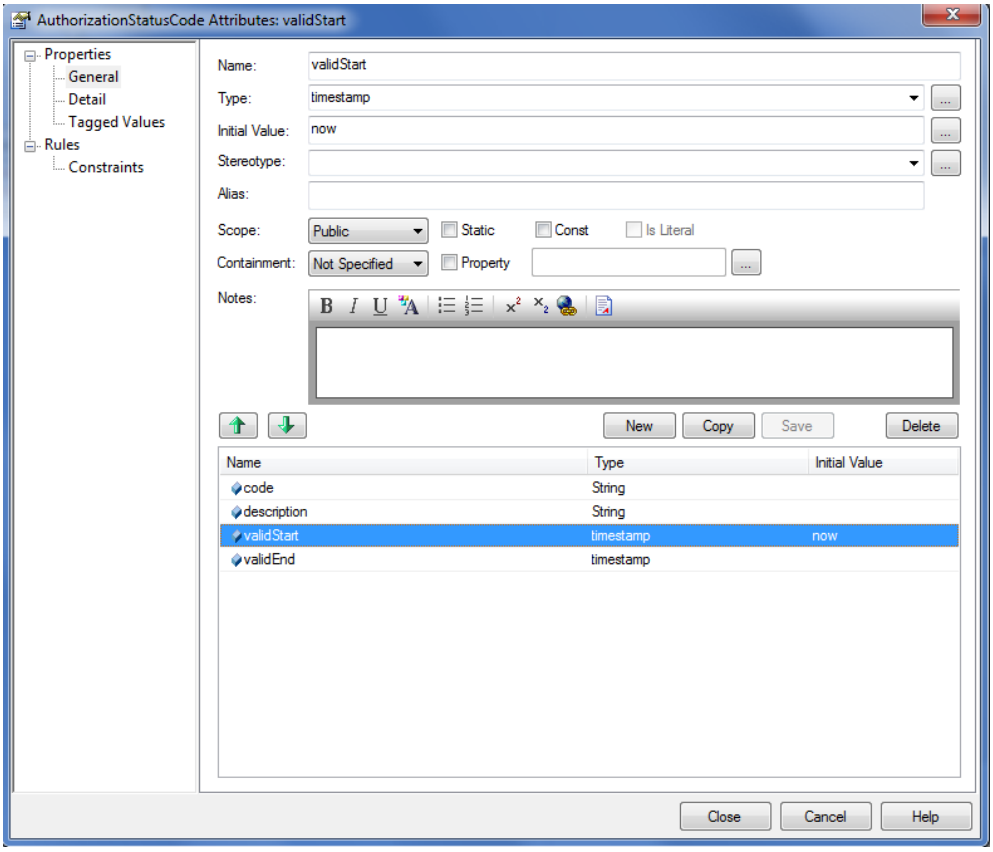
Default	The Lower bound property is defaulted to 1, which means that the corresponding column is required.
Purpose	The "Lower bound" property is used to determine whether a field is mandatory or optional. If the lower bound is set to "1", then the field is mandatory in the physical model. If the lower bound is set to "0", then the field is optional in the physical model.
Constraints	1. The "Lower bound" tag can have the values (1 | 0).

Example:



3.4.5 Attribute Default Values

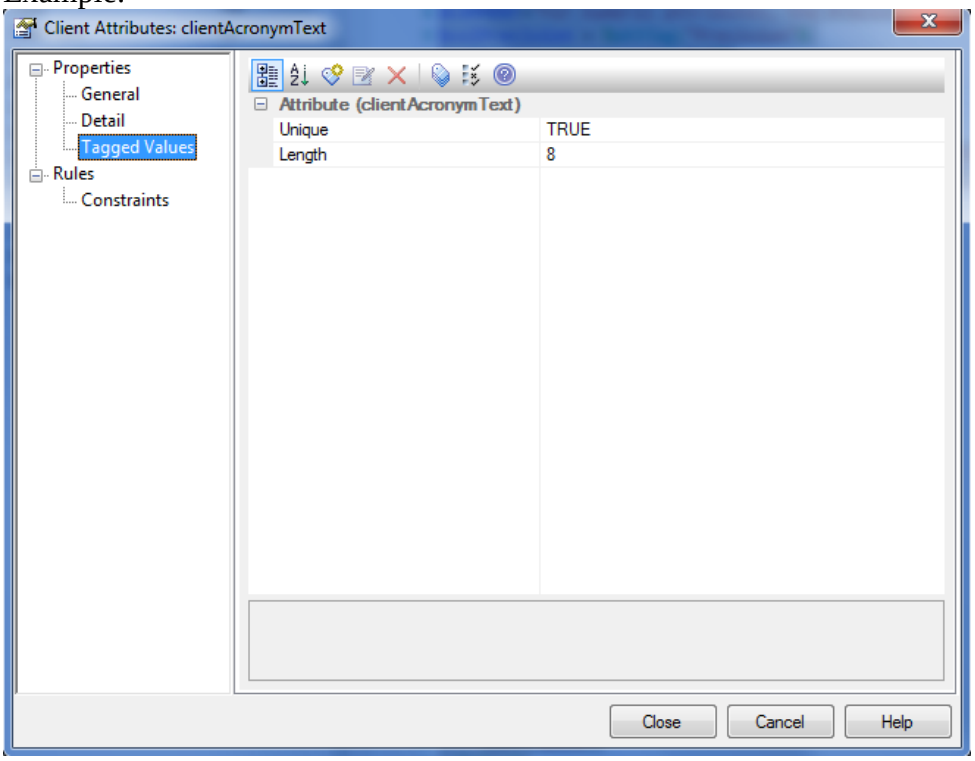
Specify an Initial Value for all relevant attributes (e.g. ValidStart).



3.4.6 Attribute Length

Name	Length
Type	Tagged Value
Required	No
Default	Defaults to a length of 4000
Purpose	The "Length" tag is used to specify the length of datatypes that support the property. The "Length" tag must be specified for the datatypes: • String The "Length" tag is standard and the "AttrLength" tag is supported historically within the logical to physical data model transformation.
Constraints	1. Only a whole number may be specified for the length.

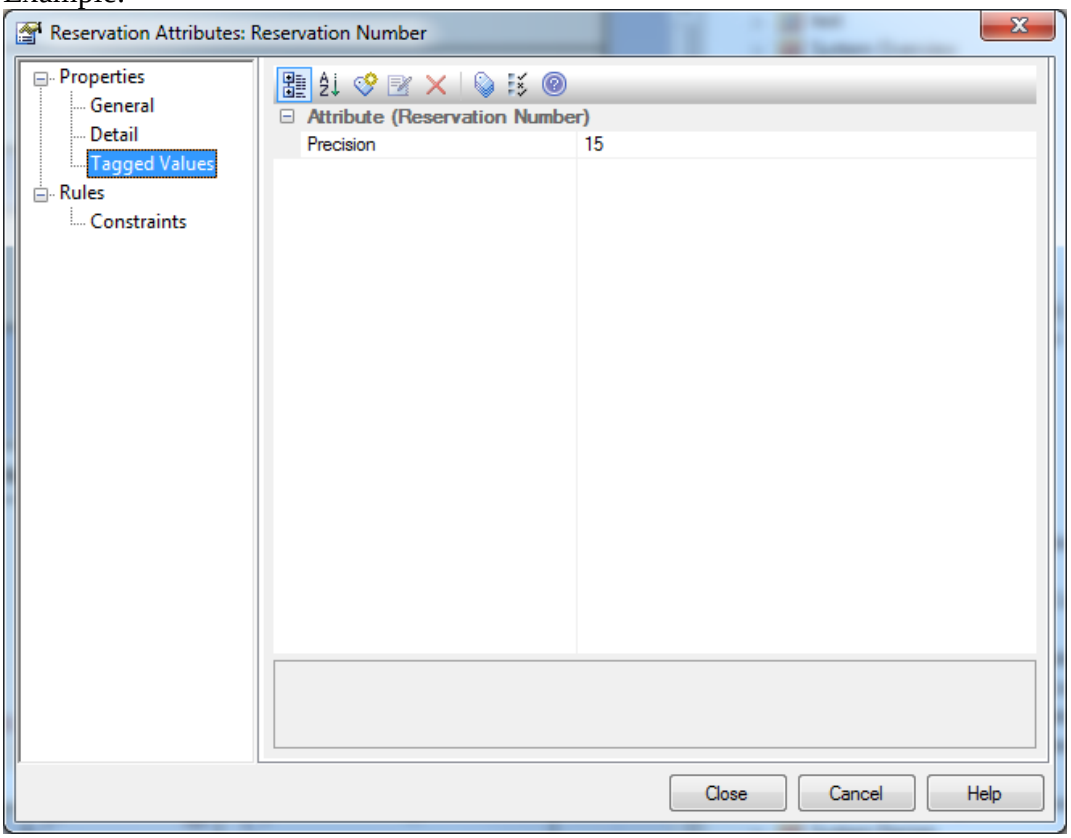
Example:



3.4.7 Attribute Precision

Name	Precision
Type	Tagged Value
Required	No
Default	Defaults to a precision of 6 for Int datatypes. Defaults to a precision of 10 for Number datatypes.
Purpose	The "Precision" tag is used to specify the precision of numeric datatypes that support the property. The "Precision" tag must be specified for the datatypes: • Number • Int The "Precision" tag is standard and the "AttrPrecision" tag is supported historically within the logical to physical data model transformation. NOTE: Model transformation treats integers with precision and numbers with just precision and no scale tag the same. Both create a physical model number data type without decimal places.
Constraints	1. Only a whole number may be specified for the precision. 2. The whole number must fall in the range 1 to 38 for Oracle numbers.

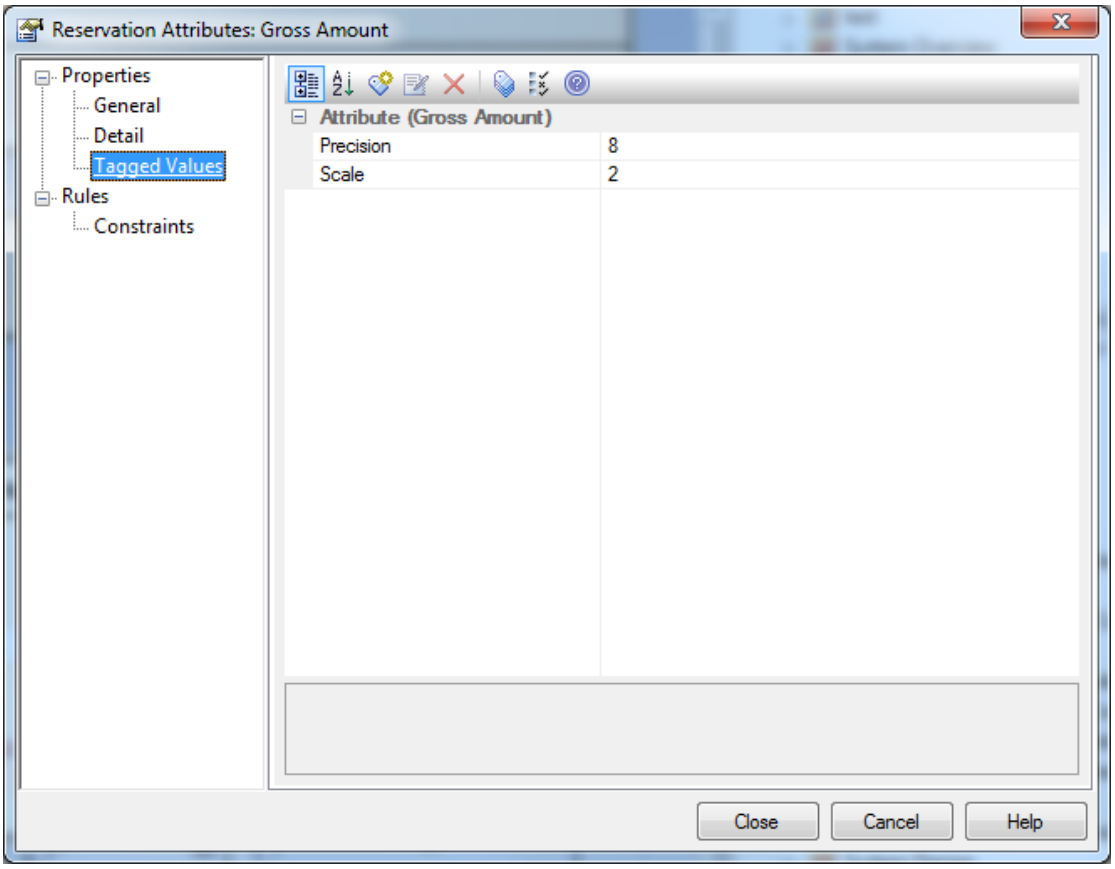
Example:



3.4.8 Attribute Scale

Name	Scale
Type	Tagged Value
Required	No
Default	Defaults to a scale of 0 for Int datatypes. Defaults to a scale of 0 for Number datatypes.
Purpose	The "Scale" tag is used to specify the number of decimal places required for numeric datatypes. The "Scale" tag must be specified for the datatypes: • Number The "Scale" tag is standard and the "AttrScale" tag is supported historically within the logical to physical data model transformation.
Constraints	1. Only a whole number may be specified for the scale. 2. The whole number must fall in the range -84 to 127 3. Scale will be ignored if the datatype is integer.

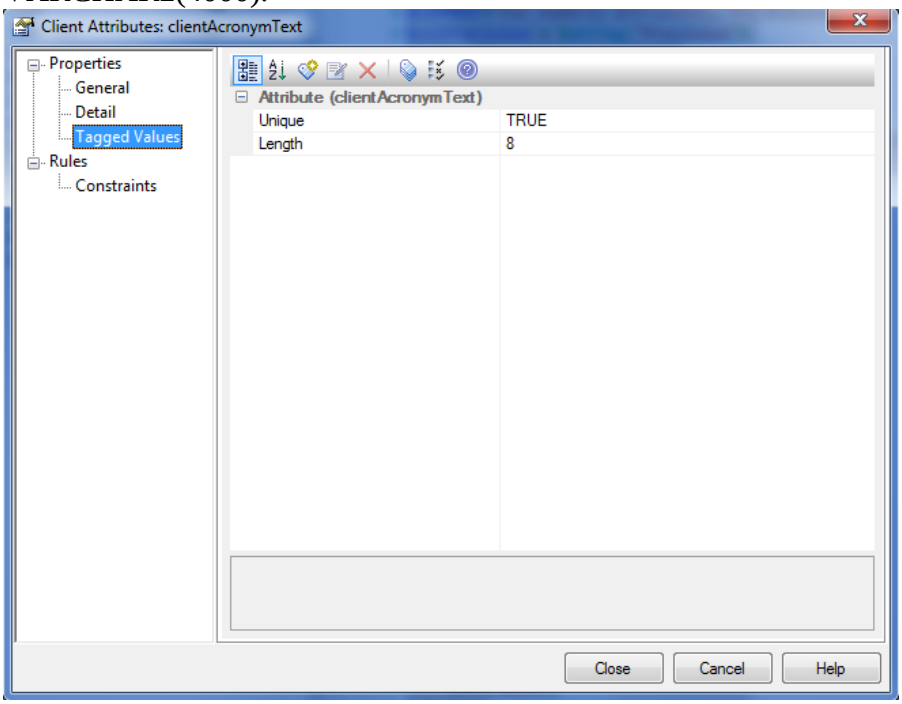
Example:


Reservation Attributes: Gross Amount
Properties
General
Detail
Tagged Values
Rules
Constraints
Attribute (Gross Amount)
Precision
8
Scale
2
Close
Cancel
Help

3.4.9 Attribute Datatypes

3.4.9.1 String Datatype

For all attributes of type “String”, add a “Length” tag on the attribute with a value equal to the string length. If this tag is not specified, all “String” attributes will be transformed to VARCHAR2(4000).



3.4.9.2 Boolean Datatype

Translates to Varchar2 length 1.
The name will end in “_IND” if not already “_INDICATOR”.
Acceptable initial values are: {true,false}which become the database default values.

3.4.9.3 Geometry or Point or BBOX Datatypes

Translate to SDO Geometry.

3.4.9.4 Date Datatype

Translates to Oracle Date.

3.4.9.5 Timestamp Datatype

Translates to Oracle TIMESTAMP length 6.

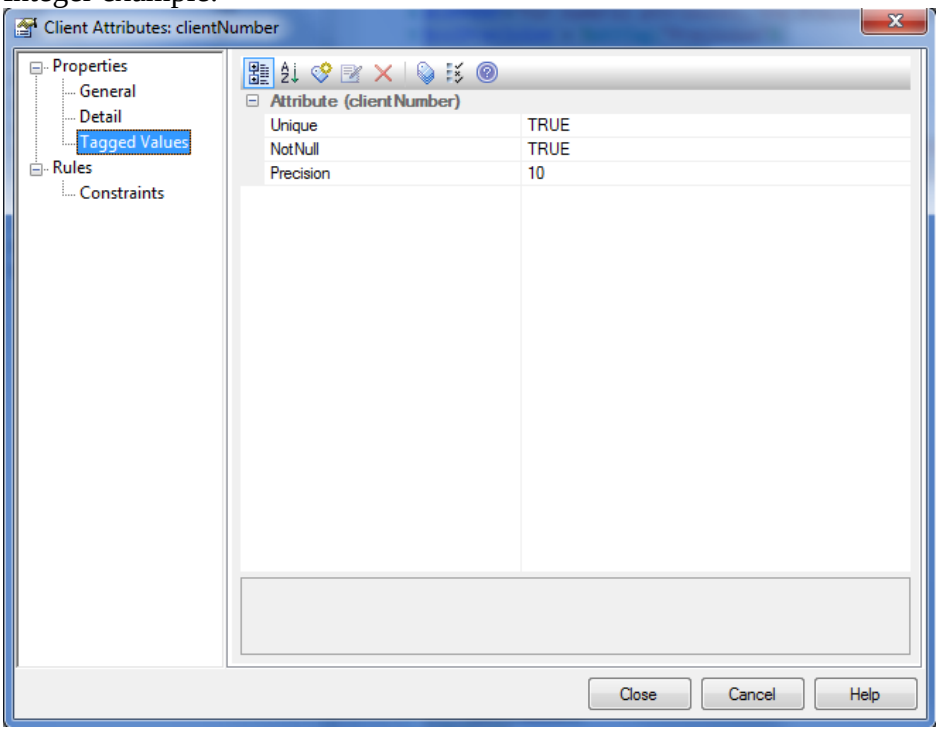
3.4.9.6 Unstructured content

Represented using BLOB or CLOB. DBA approval is required before implementing unstructured data.

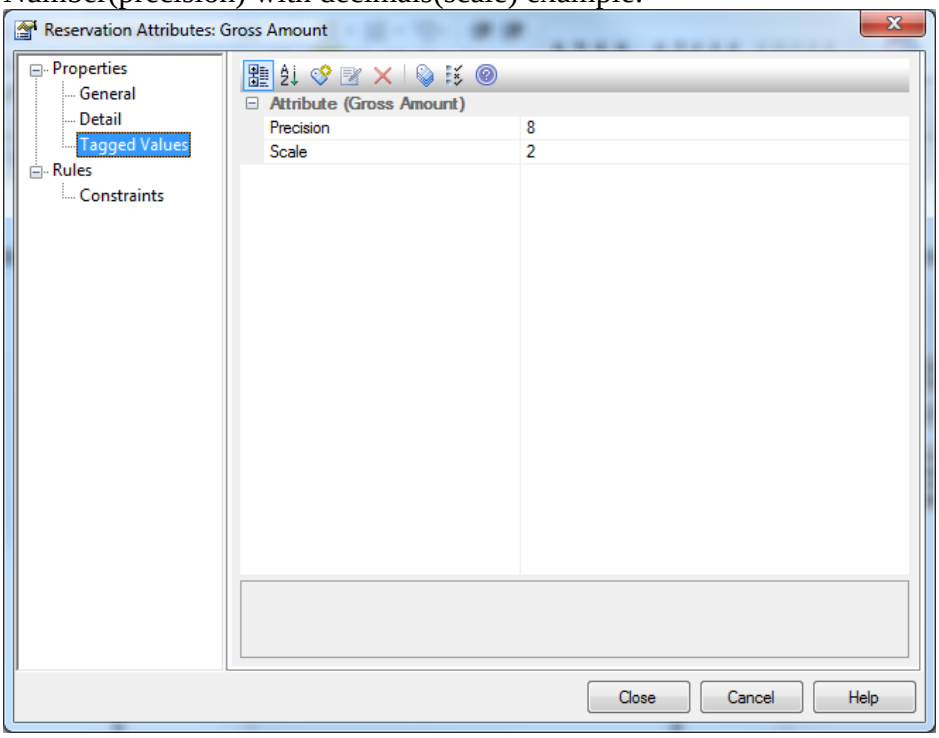
3.4.9.7 Numeric Datatype

For numeric attributes, add a “Precision” tag on the attribute. For non-integers, add both a “Precision” tag a “Scale” tag on the attribute.

Integer example:



Number(precision) with decimals(scale) example:



3.5 Association Configuration

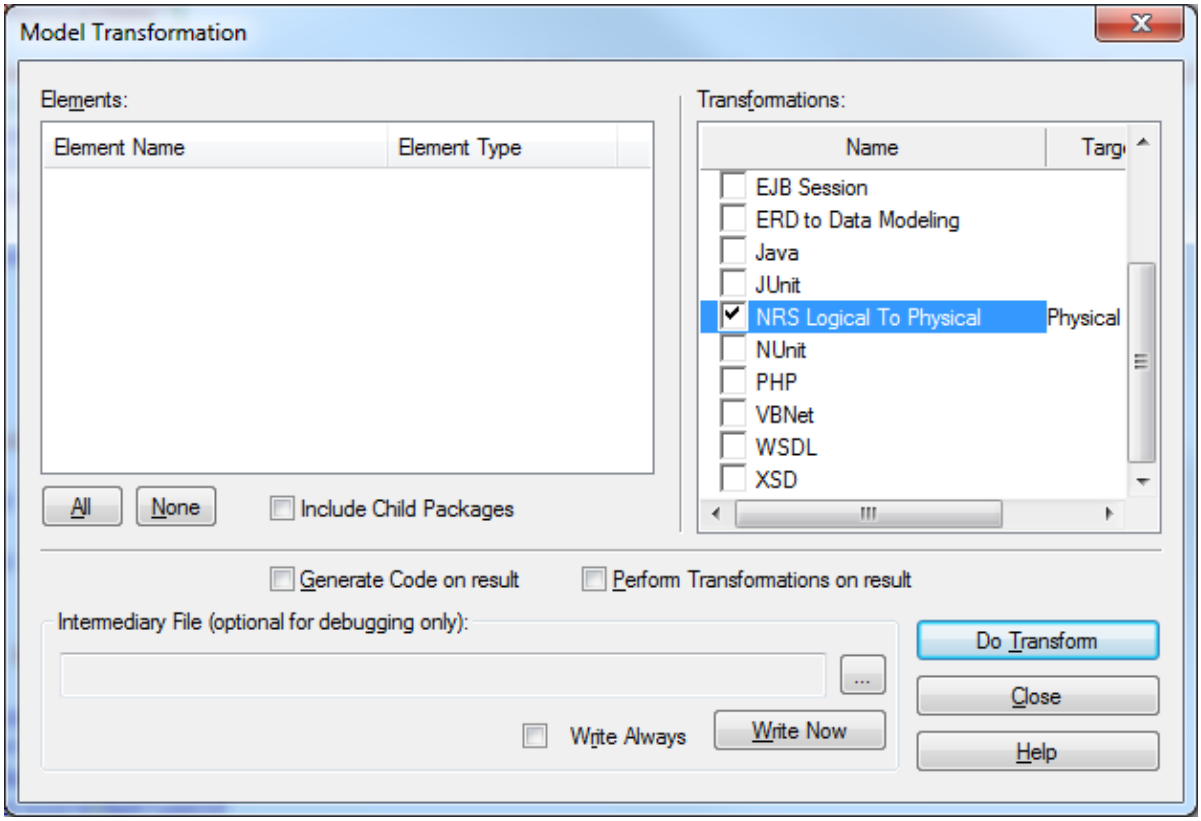
3.5.1 Self-Referencing and Multiple Association Naming

Name	PhysicalNamePrefix
Type	Tagged Value
Required	No
Default	Defaults to 'SetPNPrefix'.
Purpose	Where there is a self-referencing relationship to a class or more than one relationship from one class to another the PhysiclNamePrefix tagged value prefixes the primary key name with the PhysicalNamePrefix tagged value for the second, self-referencing, column being created. For example if there is a self referencing association from the Employee class to itself, and the employee primary key is Employee ID, a PhysicalNamePrefix of Manager would create a foreign key column of MANAGER_EMPLOYEE_ID. For example if there are two associations from Org Unit to Recreation Site and one has a PhysicalNameTag of Geographic and the other Administrative the two foreign key columns created would be GEOGRAPHIC_ORG_UNIT_ID and ADMINISTRATIVE_ORG_UNIT_ID
Constraints	Physical Name including prefix must be 30 characters or less long.

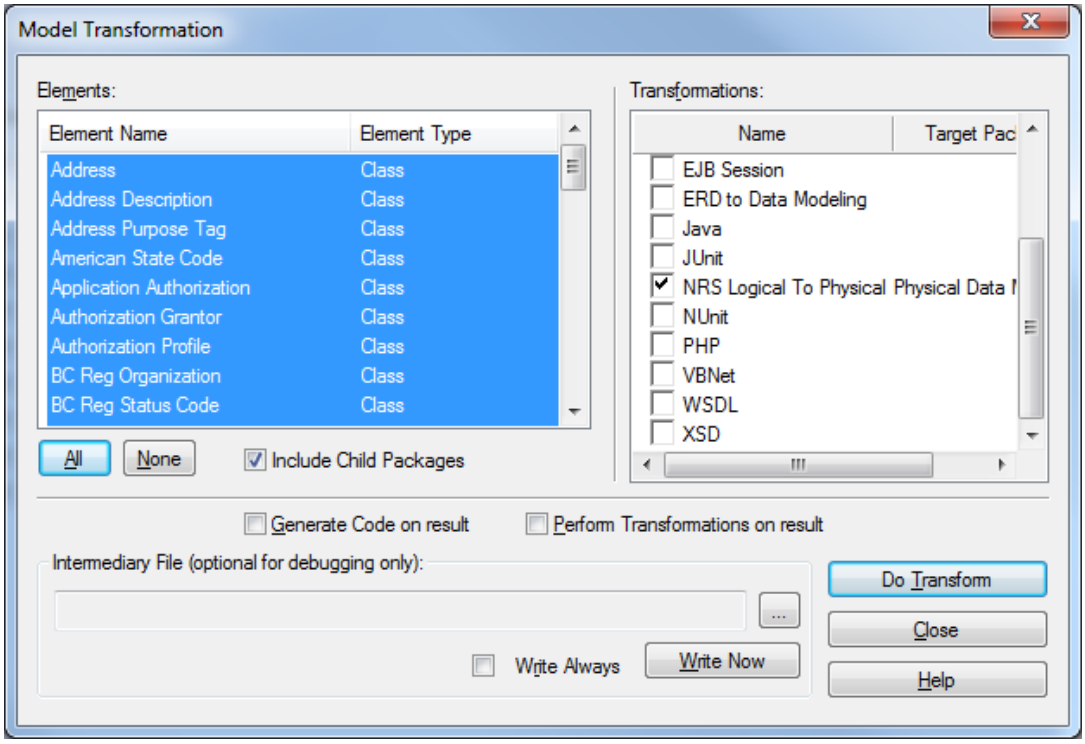
4. Logical Model to Physical Model Transformation

This section describes how to run the Model Drive Architecture (MDA) Transformation that will produce the Physical Data Model from the Logical Data Model.

- Ensure a destination package exists that will hold the transformed objects. Create a package called "Physical Data Model" to contain the target (physical) model if necessary.
- In the Project Browser, select the relevant source package (e.g. "Logical Data Model").
- Select the **Tools → Model Transformation (MDA) → Transform Current Package…** menu option
- Can also use the right click context menu on the source package, [in EA12 choose 'Advanced'], then choose 'Transform Package')



- A list of Elements (objects) will be listed on the left side. Select all or some of the tables for transformation.
- Check the "Include Child Packages" check box and ensure relevant elements are selected.
- Check the transformation named "NRS Logical To Physical". You will be prompted to select a target package for the transformed objects. Select the "Physical Data Model" package.



- Click the “Do Transform” button.

5. Physical Model Preparation

5.1 Post-Transformation Modifications to Physical Database Model

The following steps need to be completed prior to generating DDL.

5.1.1 Review Tables

- For each table:
    - Ensure columns are in the right physical model column order as defined in the Physical Model standards
- While the logical model QA should resolve any names over 30 characters through a name length check and PhysicalName tag for any attributes over 30 characters, if any names are over 30 characters during the transformation they will be truncated which may not be what is required. Check to ensure there are no table names greater than 30 characters, or names have been truncated.
-

5.1.2 Review Columns

- Shorten any column names that are greater the 30 characters.
- For sequence configuration
    - For primary key type ID columns the AutoNum column property is set to True.
    - For other columns that require a sequence set the AutoNum column property to True.
    - The Oracle DDL transformation template available in the NRS Logical to Physical XMI reference data import file will name sequences to ministry standards. If not used, please note that the sequence and trigger names will have to be manually changed after the DDL is generated to meet physical naming standards.
- Review all columns that were generated from boolean attributes in the logical model. The custom attribute template already adds a default value if the initial value of the attribute in the logical model is true or false, but this wasn't specified in the logical model then set the default column value to 'Y' or 'N' (including quotes) if appropriate.
- Add any initial values not defined on the logical model. We should probably define initial values on the logical model so the values get brought across by the DDL Transformation (e.g. an initial value of "now" on Effective Date attributes is transformed by the custom attribute template to default value of SYSDATE).

5.1.3 Review Foreign Keys

- For each foreign key:
    - Confirm that the correct columns have been included (GUID, ID, CODE, LIST, Business Key).

- For each foreign key column, set it to NOT NULL (if appropriate). The DDL Transformation does not do this by default (currently).

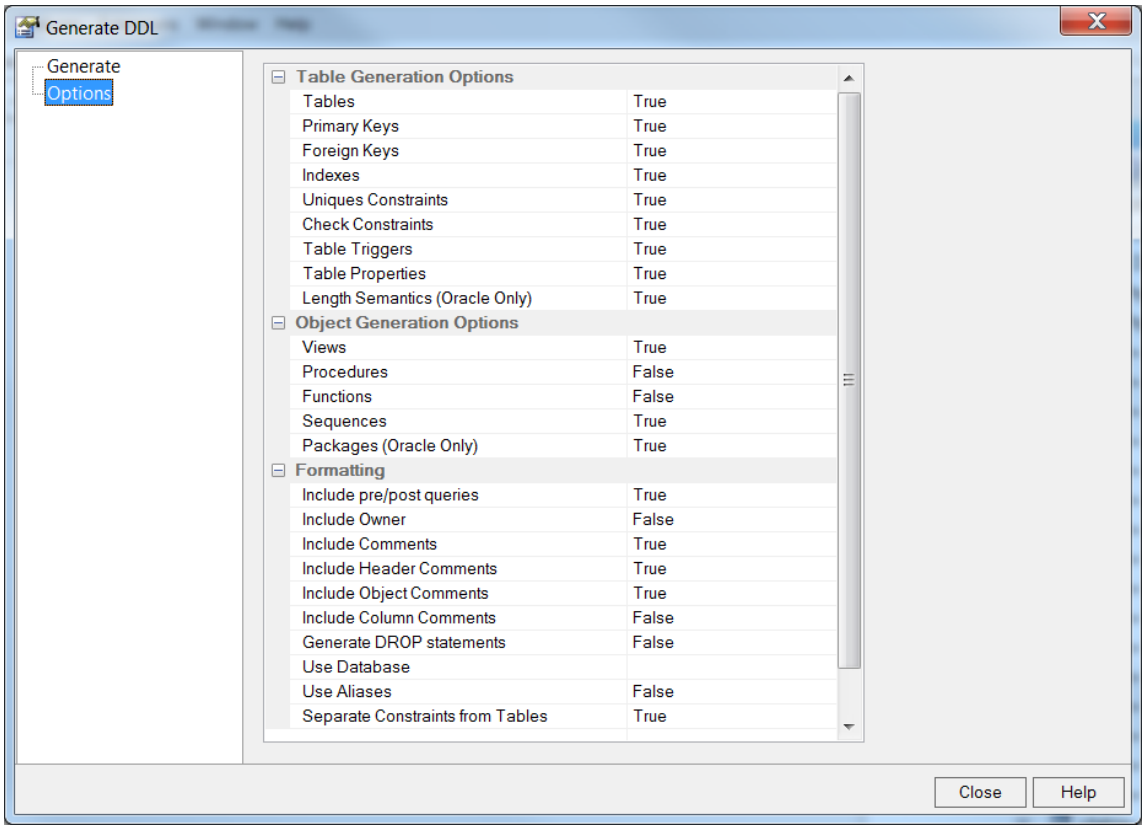
5.1.4 Review Indexes

- Unique indexes are created through transformation from logical to physical from the business key set up in the logical model (stereotypes and tagged values). Further indexes may be created in the physical model if needed.
- For each index that is required:
    - Right mouse click on the appropriate table and select *Features & Properties* → *Operations…*
    - Type in the appropriate (unique) index name and then select a Stereotype of "index".
    - Select the "Column" tab (under Properties) and select the appropriate column(s).

6. Physical Model to DDL Script Generation

This section provides steps for generating Oracle DDL Scripts from the Physical Data Model.

- Select the package containing the physical model in the Project Browser.
- In EA 10 Select ***Tools → Database Engineering → Generate Package DDL*** menu option.
- In EA 12 Select ***Package → Database Engineering → Generate Package DDL*** menu option.



- Select the appropriate options.
- Press the *Generate* button.

6.1 Post-Generation Changes to DDL

It is likely that some manual changes will need to be made to the DDL. Therefore, the generated DDL file(s) should be placed under version control.

- If any of the tables include spatial columns (e.g. columns of type SDO_GEOMETRY) then post-generation DDL changes will be required. That is because Enterprise Architect cannot generate DDL to create indexes other than normal B-Tree indexes. In addition, Enterprise Architect cannot generate INSERT statements to populate spatial metadata For instance,

DDL similar to the following must be manually created for each column of type SDO_GEOMETRY:

```
INSERT INTO USER_SDO_GEOM_METADATA
  (TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
VALUES
  ('TA_PARK_ECORES_SMVW', 'GEOMETRY',
   MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X', 200000, 1900000, 0.0005),
                       MDSYS.SDO_DIM_ELEMENT('Y', 300000, 1800000, 0.0005)),
   3005);

CREATE INDEX TASMVW_SP_SP_IDX ON TA_PARK_ECORES_SMVW (GEOMETRY)
   INDEXTYPE IS MDSYS.SPATIAL_INDEX PARAMETERS ('sdo_dml_batch_size=1');
```

- Enterprise Architect cannot generate DDL for indexes other than the run-of-the-mill B-Tree indexes. An enhancement request to Sparx Systems has been submitted for this. The tool should be able to generate bitmap, bitmap join and spatial indexes, but currently cannot. The CREATE INDEX statements will have to be manually modified for these other types of indexes until this enhancement request is implemented.
- If any of the tables contain valid time columns (VALID_START and VALID_END) then it is necessary to generate and run ALTER TABLE statements to create the valid time period pseudo-column. That is because Enterprise Architect does not currently support the "PERIOD FOR" pseudo-column (I have, however, submitted an enhancement request to Sparx Systems). To create these hidden columns:
    - Connect to the database using the schema that owns the tables.
    - Run the SELECT statement contained in *Appendix B - Generating DDL for Valid Time Period Pseudo-columns*.
    - Run the resulting ALTER TABLE statements to create the valid time period pseudo-columns.

7. Appendix A – Contents of the NRS Logical to Physical Transformation Templates

7.1 Transformation Reference Data

As noted in Section 2.3 - **NRS DDL Transformation Templates**, a custom NRS Logical to Physical XMI reference data import file has been created to import custom templates which support transformation of logical data models to physical database models targeted for implementation in an Oracle database.

The following sections describe the NRS custom transformation code.

7.2 File Template

Identifies the version of the transformation templates, describes what the transformation does and creates an NRS Physical Model package for the physical data model. The File Template is the starting template for the transformation with other templates called, using the **list** function, to complete the components of the transformation.

7.3 Class Template

The Class Template is used to transform classes in the Logical Model to tables in the Physical Model. It:

- sets the physical name for the tables from the PhysicalName tag if one exists for the class
- creates business tables and list of values (list and code) tables
- creates the primary key based on the PrimaryKeyType tag in the logical model class
- creates the list of columns defined in the class attributes unless it is a list or code table which have standard attributes
- adds a set of standardized audit columns
- creates column unique indexes based on attribute business key stereotypes and tagged values

7.4 Attribute Template

The Attribute Template is used to transform table attributes from the Logical Model to columns in the Physical Model. It:

- creates the list of columns from the class attributes
- sets column data types, lengths, optionality
- sets the physical name for the columns from the PhysicalName tag if the attribute has one

7.5 Connector Template

The Connector Template used is to transform associations from the Logical Model to foreign keys in the Physical Model. It creates foreign key constraints based on logical model associations.

7.6 Primary Key Template

This section contains the script for a custom template called “Attribute__Primary Key” The template is used to generate manually defined primary keys from the Logical Model to the Physical Model.

7.7 Other Templates

New templates have been added for calls to support business key, unique key and foreign key creation.

8. Appendix B – Generating DDL for Valid Time Period Pseudo-columns

Enterprise Architect does not support the definition of PERIOD FOR pseudo-columns within physical models. It is therefore not possible to generate the following DDL out of the tool:

```
CREATE TABLE CCL_ALTERNATE_ORG_NAME_TYPES
(
     NAME_TYPE_CODE          VARCHAR2(5) NOT NULL,
     NAME_TYPE_DESCRIPTION   VARCHAR2(100) NOT NULL,
     VALID_START             TIMESTAMP(6) DEFAULT
SYSTIMESTAMP NOT NULL,
     VALID_END               TIMESTAMP(6) NULL,
     PERIOD FOR VALID_TIME (VALID_START, VALID_END)
      …
);
```

However, the following select statement can be used to generate ALTER TABLE statements to create PERIOD FOR pseudo-columns for all tables that make use of valid time:

```
SELECT 'ALTER TABLE ' || TABLE_NAME ||
       ' ADD PERIOD FOR VALID_TIME (VALID_START,
VALID_END);'
FROM USER_TAB_COLS
WHERE COLUMN_NAME = 'VALID_START'
ORDER BY TABLE_NAME;
```

Please note that the code above assumes that the VALID_START and VALID_END columns have already been created in all tables owned by the current schema that require valid time.

9. Appendix C – Planned Changes to Logical to Physical Transformation

There are some changes which will improve the functionality of the. Future releases of Enterprise Architect might better support desired improvement in transformation functionality. Further work on the transformation scripts may allow NRS to achieve some desired functionality.

Some planned changes to the logical to physical data model transformation scripts are anticipated.

- suggestion to handle table grants within EA and transformation
- ability to pass parameters and simplify transformation templates
- suggestion to capture table size and annual growth
- customization of new DDL transformation template (New to EA12) to streamline DDL generation