How To Learn Data Abstraction Best Practices

An Open Source Asset for use with TIBCO® Data Virtualization

Project Name	AS Assets Data Abstraction Best Practices
Document Location	This document is only valid on the day it was printed. The source of the document will be found in the ASAssets_DataAbstractionBestPractices folder (https://github.com/TIBCOSoftware)
Purpose	Self-paced instructional

Revision History

Version	Date	Author	Comments
1.0	07/18/2011	Mike Tinius	Initial revision
2.0	8/30/2013	Mike Tinius	Updated format.
8.0	12/05/2013	Mike Tinius	Updated for Best Practices v8.0
8.1	2/21/2014	Mike Tinius	Updated for Best Practices v8.1
8.1.2	5/12/2014	Mike Tinius	Updated for Best Practices v8.1.2 – Rebrand PS Assets to AS Assets.
8.1.3	8/8/2014	Mike Tinius	Updated for Best Practices v8.1.3 – New text to review for generateCast variable.
8.1.4	08/25/2014	Mike Tinius	Updated for Best Practices v8.1.4
8.1.5	11/26/2014	Mike Tinius	Updated for Best Practices v8.1.5
8.1.6	05/20/2015	Mike Tinius	Updated for Best Practices v8.1.6 – Powerpoint format only.
8.1.7	09/21/2015	Mike Tinius	Updated for Best Practices v8.1.7 – added generateViews=2 to allow generating views with a SELECT * projection.
8.1.8	05/24/2017	Mike Tinius	Updated for Best Practices v8.1.8 – added Privilege scripts.
8.1.9	12/06/2017	Mike Tinius	Transitioned to Tibco for release 8.1.9
2018Q1	03/20/2018	Mike Tinius	Release 2018Q1 – no changes.

Related Documents

Name	Version
How To Use Utilities.pdf	2018Q1
How To Use Data Abstraction Best Practices.pdf	2018Q1
How To Test Data Abstraction Best Practices.pdf	2018Q1
How To Use Data Abstraction Best Practices Manage Annotations.pdf	2018Q1
How To Use Data Abstraction Best Practices Privilege Scripts.pdf	2018Q1

Supported Versions

Name	Version
TIBCO® Data Virtualization	7.0 or later
AS Assets Utilities open source	2018Q1 or later

Table of Contents

1 Practice Goals

We will use the Data Abstraction Best Practices to create a project structure and generate views.

2   Overview

The Data Abstraction Best Practices provide a template for creating a project according to the layered approach. The Best Practices generation scripts are used to generate the different layers of the Data Abstraction Best Practices. For additional information, please refer to the document "**How To Use AS Data Abstraction Best Practices.pdf**".

3   Pre-Requisites

Composite examples "/shared/examples" must exist.

Instructor creates /shared/labs and students receive a lab number…e.g. lab01, lab02 etc.

Instructor installs Composite Data Abstraction Best Practices for the class

I   Install and Deploy the Data Abstraction Best Practices Framework

Refer to the "**Installation**" section in the "**How To Use DVBU AS Best Practices.pdf**" for complete instructions on installing the Data Abstraction Best Practices scripts and Utilities.

4   Lab Procedures

2 Create a new project "/labs/lab##" from the Best Practices template.

CREATE PROJECT [AUTOMATED]

Follow the steps below to create a new project.

DIRECTIONS:

1. **Generate and Configure project** – Configure a new project
   a. Expand the folder
      /shared/ASAssets/BestPractices_vXX/_ProjectMaintenance
      i. Open **generateProject**(projectPath, generateTestFolder)
   b. Click Execute [image] and enter the following parameters
      i. projectPath= **/shared/labs/lab##**
         1. replace ## with your lab id…e.g. lab00
      ii. generateTestFolder=<you choose>
         1. 1=yes, generate – this options is for the school of thought who want to keep all of their test views and scripts in a separate, mirror structure to the BestPractices structure.
         2. 0=no, do not generate – this option is for the school of thought who don't want a separate mirror structure but prefer to create test sub-folders within the main BestPractices structure.
      iii. overwrite=0
         1. 1=yes, overwrite the existing project if it exists.
         2. 0=no, do no overwrite the existing project if it exists.
   c. A message appears such as:
      i. Project [/shared/labs/lab##] successfully configured. Click the refresh button in Studio.
   d. Click refresh [image] when the procedure finishes to refresh Studio.

BACKGROUND:

2. **Note:** – this procedure automatically performs the following:
   a. Copies the template folder "DataAbstraction_GENERIC_Template" to the path you specify.

    b. Modifies the "basePath" variable in /shared/labs/lab##/_scripts/Constants/defaultValues.

    c. Rebinds several procedures to point to /shared/labs/lab## resources instead of the default template folder "DataAbstraction_GENERIC_Template".

    d. Update /Documentation trigger parameter paths

    e. Verify paths have been updated

    f. Generate the Test folder if the user requested it

3 Setup a sample data source.

DIRECTIONS:

1. Open /shared/ASAssets/BestPractices_vXX/DataAbstractionSample/Physical

2. Copy the folder "Metadata"

3. Paste into /shared/labs/lab##/Physical overwriting the existing "Metadata" folder

4. Delete the folder: /Physical/Metadata/OracleSource

4 Create an XML to Relational Transformation

OBJECTIVE:

1. In this section of the lab, you will create a XML to relational transformation which the Formatting view for the "XML" group will be based off of. This is done to demonstrate how Formatting views can be generated from procedures including XSLT, Parameterized Queries, and Custom SQL Script Procedures with a cursor and Packaged Queries with a cursor output.

DIRECTIONS:

2. **Create a folder** called "**ds_XML**" under /Physical/Formatting/Transformations

3. Right-click on "ds_XML" and create a "New Transformation" as an XSLT procedure
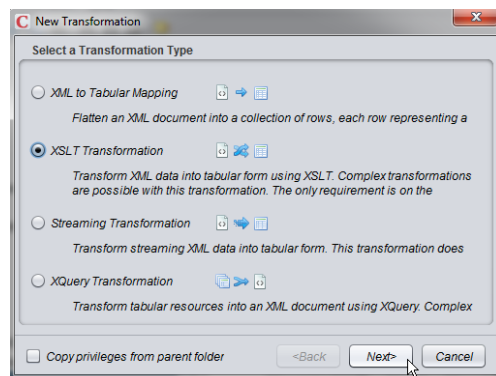
4. Right-click on "ds_XML"

    a. Create a new "XSLT Transformation" called **productCatalog_Transformation**

        i. Enter the name in Transformation Name

        ii. Browse to the Physical XML data source and select it: /shared/labs/lab##/Physical/Metadata/XML/ds_XML/productCatalog.xml

        iii. Click Finish
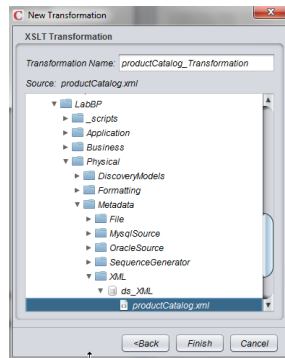
5. Select CategoryID and CategoryName and click the "Create Link And Target" button
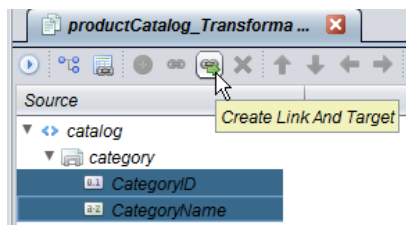
6. Highlight the remaining fields and click the "Create Link And Target" button
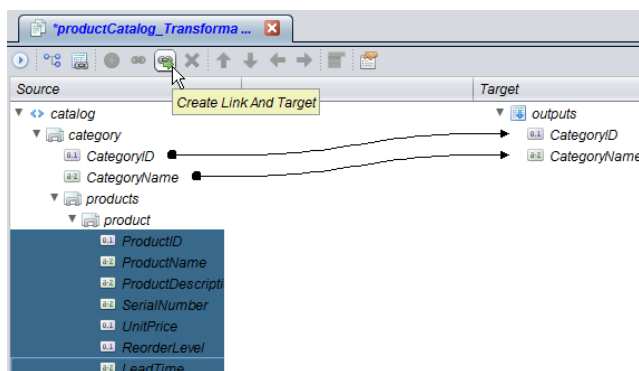
7. Save and execute

5 Generate Configuration Starting Folders

CONFIGURE STARTING FOLDERS [AUTOMATED]
Follow the steps below to generate the ConfigureStartingFolders.

OBJECTIVE:

The ConfigureStartingFolders script can either be generated or created from hand. This part of the lab focuses on generating the procedure using the "generateConfigureStartingFolders()".

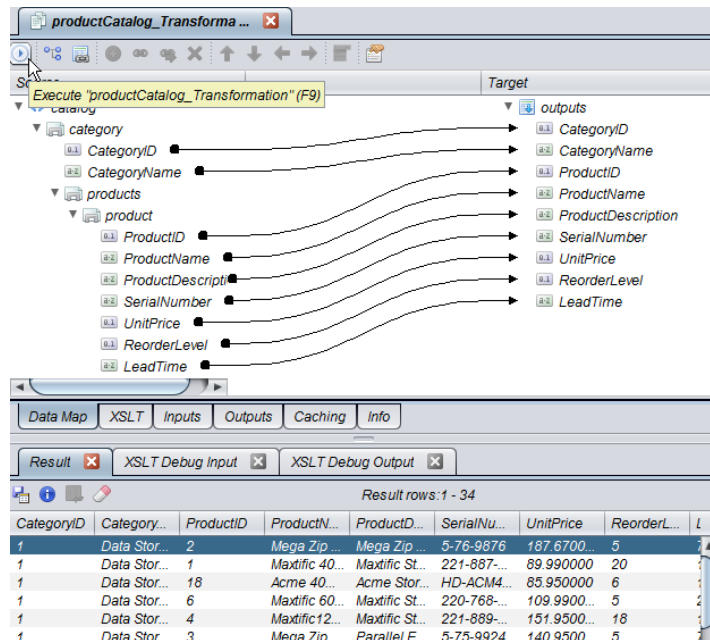*Note*: These folders are the key to the generation scripts. The entries tell the generation scripts which source folders and target folders to use for the generation. There is an "INSERT" template for each level of the Best Practices containing a source and target folder. Typically, the target folder for one level becomes the source folder for the next level up. Modify the folders according to the sources that you have and your folder structure. The ConfigureStartingFolders can be generated automatically or edited by hand.

This procedure is used to generate the ConfigureStartingFolders() procedure based on data sources and transformations found in both the /Physical/Metadata and /Physical/Formatting/Transformations folders.

DIRECTIONS:

1. **Create "ConfigureStartingFolders" procedure**

   i. Expand the folder:
      /shared/ASAssets/BestPractices_vXX/_ProjectMaintenance

      1. Open **generateConfigureStartingFolders**(projectPath)

    ii. Click Execute [icon] and enter the following parameters
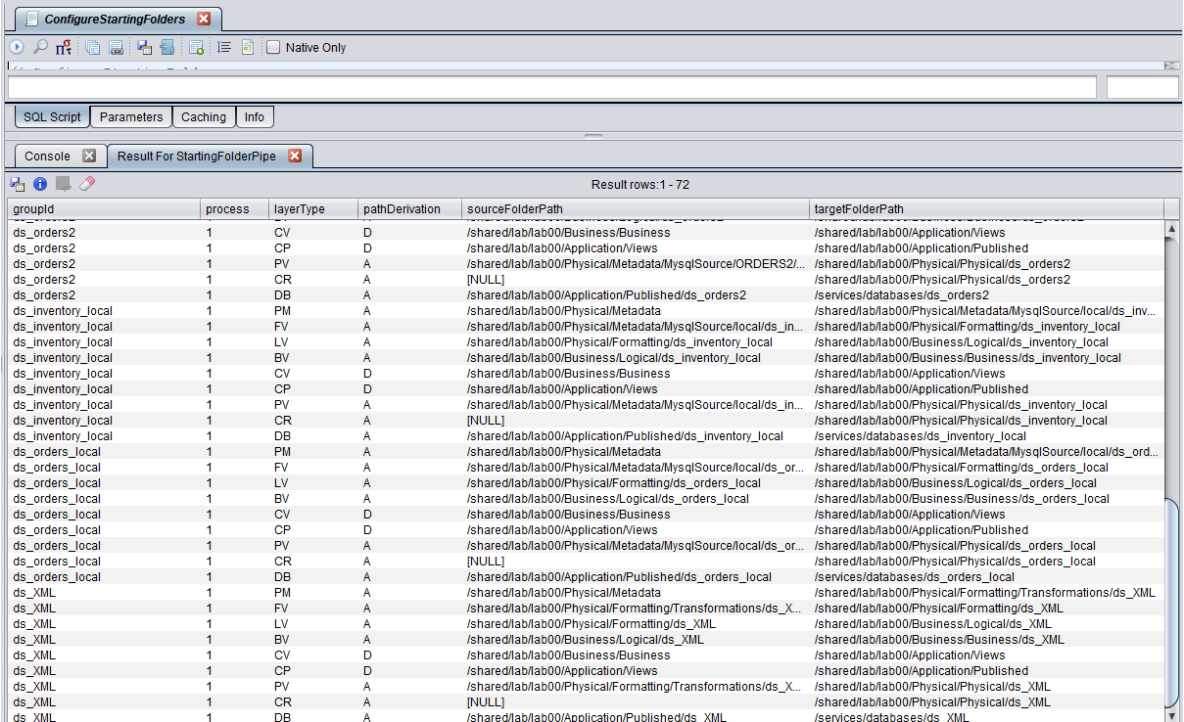
        1. projectPath= **/shared/labs/lab##**

        2. replace ## with your lab id…e.g. lab00

    iii. Click refresh [icon] when the procedure finishes to refresh Studio.

2. Expand the folder /shared/labs/lab##/_scripts/Configure

    a. Open ConfigureStartingFolders

    b. Click Execute [icon] to see the results as shown below

groupId	process	layerType	pathDerivation	sourceFolderPath	targetFolderPath
ds_orders2	1	CV	D	/shared/lab/lab00/Business/Business	/shared/lab/lab00/Application/Views
ds_orders2	1	CP	D	/shared/lab/lab00/Application/Views	/shared/lab/lab00/Application/Published
ds_orders2	1	PV	A	/shared/lab/lab00/Physical/Metadata/MysqlSource/ORDERS2/...	/shared/lab/lab00/Physical/Physical/ds_orders2
ds_orders2	1	CR	A	[NULL]	/shared/lab/lab00/Physical/Physical/ds_orders2
ds_orders2	1	DB	A	/shared/lab/lab00/Application/Published/ds_orders2	/services/databases/ds_orders2
ds_inventory_local	1	PM	A	/shared/lab/lab00/Physical/Metadata	/shared/lab/lab00/Physical/Metadata/MysqlSource/local/ds_inv...
ds_inventory_local	1	FV	A	/shared/lab/lab00/Physical/Metadata/MysqlSource/local/ds_in...	/shared/lab/lab00/Physical/Formatting/ds_inventory_local
ds_inventory_local	1	LV	A	/shared/lab/lab00/Physical/Formatting/ds_inventory_local	/shared/lab/lab00/Business/Logical/ds_inventory_local
ds_inventory_local	1	BV	A	/shared/lab/lab00/Business/Logical/ds_inventory_local	/shared/lab/lab00/Business/Business/ds_inventory_local
ds_inventory_local	1	CV	D	/shared/lab/lab00/Business/Business	/shared/lab/lab00/Application/Views
ds_inventory_local	1	CP	D	/shared/lab/lab00/Application/Views	/shared/lab/lab00/Application/Published
ds_inventory_local	1	PV	A	/shared/lab/lab00/Physical/Metadata/MysqlSource/local/ds_in...	/shared/lab/lab00/Physical/Physical/ds_inventory_local
ds_inventory_local	1	CR	A	[NULL]	/shared/lab/lab00/Physical/Physical/ds_inventory_local
ds_inventory_local	1	DB	A	/shared/lab/lab00/Application/Published/ds_inventory_local	/services/databases/ds_inventory_local
ds_orders_local	1	PM	A	/shared/lab/lab00/Physical/Metadata	/shared/lab/lab00/Physical/Metadata/MysqlSource/local/ds_ord...
ds_orders_local	1	FV	A	/shared/lab/lab00/Physical/Metadata/MysqlSource/local/ds_or...	/shared/lab/lab00/Physical/Formatting/ds_orders_local
ds_orders_local	1	LV	A	/shared/lab/lab00/Physical/Formatting/ds_orders_local	/shared/lab/lab00/Business/Logical/ds_orders_local
ds_orders_local	1	BV	A	/shared/lab/lab00/Business/Logical/ds_orders_local	/shared/lab/lab00/Business/Business/ds_orders_local
ds_orders_local	1	CV	D	/shared/lab/lab00/Business/Business	/shared/lab/lab00/Application/Views
ds_orders_local	1	CP	D	/shared/lab/lab00/Application/Views	/shared/lab/lab00/Application/Published
ds_orders_local	1	PV	A	/shared/lab/lab00/Physical/Metadata/MysqlSource/local/ds_or...	/shared/lab/lab00/Physical/Physical/ds_orders_local
ds_orders_local	1	CR	A	[NULL]	/shared/lab/lab00/Physical/Physical/ds_orders_local
ds_orders_local	1	DB	A	/shared/lab/lab00/Application/Published/ds_orders_local	/services/databases/ds_orders_local
ds_XML	1	PM	A	/shared/lab/lab00/Physical/Metadata	/shared/lab/lab00/Physical/Formatting/Transformations/ds_XML
ds_XML	1	FV	A	/shared/lab/lab00/Physical/Formatting/Transformations/ds_X...	/shared/lab/lab00/Physical/Formatting/ds_XML
ds_XML	1	LV	A	/shared/lab/lab00/Physical/Formatting/ds_XML	/shared/lab/lab00/Business/Logical/ds_XML
ds_XML	1	BV	A	/shared/lab/lab00/Business/Logical/ds_XML	/shared/lab/lab00/Business/Business/ds_XML
ds_XML	1	CV	D	/shared/lab/lab00/Business/Business	/shared/lab/lab00/Application/Views
ds_XML	1	CP	D	/shared/lab/lab00/Application/Views	/shared/lab/lab00/Application/Published
ds_XML	1	PV	A	/shared/lab/lab00/Physical/Formatting/Transformations/ds_X...	/shared/lab/lab00/Physical/Physical/ds_XML
ds_XML	1	CR	A	[NULL]	/shared/lab/lab00/Physical/Physical/ds_XML
ds_XML	1	DB	A	/shared/lab/lab00/Application/Published/ds_XML	/services/databases/ds_XML

Result rows:1 - 72

    a. Close the results tab

BACKGROUND:

3. Review Variable Declarations

    a. Notice how the variable declaration section resolves the base paths to the "defaultValues" constants. This makes the project directory easier to move and rename in the future since paths are derived from the "basePath" variable in "defaultValues". Variables shown below

```
-----------------------------------------------------------------
-- VARIABLE DECLARATIONS
-----------------------------------------------------------------
```

```
-- Default base folder locations defined in /Constants/defaultValues
-- (It makes maintenance easier if the base project folder is moved)

DECLARE physicalMetadataPath VARCHAR(1024) DEFAULT
/shared/labs/lab##/constants/defaultValues. physicalMetadataPath;

DECLARE physicalFormattingPathVARCHAR(1024) DEFAULT
/shared/labs/lab##/constants/defaultValues. physicalFormattingPath;

DECLARE businessLogicalPath       VARCHAR(1024) DEFAULT
/shared/labs/lab##/constants/defaultValues. businessLogicalPath;

DECLARE applicationViewsPath      VARCHAR(1024) DEFAULT
/shared/labs/lab##/constants/defaultValues. applicationViewsPath;
```

1. Review Group Identifier section for **ds_orders1**

   a. The group identifier provides the user with the ability to place multiple insert statements into a grouping. Later, this will become a useful parameter filter when generating the views. It will allow the user to specify which group or groups to generate views for.

      ```
      set groupId = 'ds_orders1';
      ```

2. Review the structure. Review how the Source and Target variables reference the path variables.

   a. This example shows how to specify the Physical Metadata CIS data source path for MySQL. Specify the full path all the way down to the data source or schema folder. This is the folder just above the tables.

   b. For an Oracle path, the physical metadata would be concatenated to the end and reference the Oracle schema.

   ```
   set groupId = 'ds_orders1';

   SET PM_FOLDER=physicalMetadataPath||'/MysqlSource/ORDERS1/ds_orders1';

   SET PV_FOLDER=physicalViewsPath||'/ds_orders1';

   SET FV_FOLDER=physicalFormattingPath||'/ds_orders1';

   SET LV_FOLDER=businessLogicalPath||'/ds_orders1';

   SET BV_FOLDER=businessBusinessPath||'/ds_orders1';

   SET CV_FOLDER=applicationViewsPath||'/ds_orders1';

   SET CP_FOLDER=applicationPublishedPath||'/ds_orders1';

   SET DB_FOLDER=compositeDatabasePath||'/ds_orders1';
   ```

3. Review how the insert statements have the target folder for one level is the source folder for the next level up. Consider the levels visually:

   - DB=Database published   (Published Database)

   - CR=Create,Read,Update,Delete   (CRUD Views)

   - CP = Application/Published   (Client Published)

- CV = Application/Views　　　(Client Views)
- BV=Business Views　　　　　(Business – Business Views
- LV = Logical Views　　　　　(Business – Logical Views)
- FV = Physical/Formatting　　(Formatting Views)
- PV=Physical Layer　　　　　(Physical Views)
- PM=Physical Metadata　　　(Physical Metadata Tables)

-- Generate Composite Database views from the client published views

INSERT INTO StartingFolderPipe VALUES (groupId,1,'DB','A',CP_FOLDER,DB_FOLDER);

-- Generate Physical_Metadata source path specified

INSERT INTO StartingFolderPipe VALUES (groupId,1,'PM','A',physicalMetadataPath,PM_FOLDER);

-- Generate Formatting_Views from the Physical_Metadata source path specified

INSERT INTO StartingFolderPipe VALUES (groupId,1,'FV','A',PM_FOLDER,FV_FOLDER);

-- Generate Logical_Views from the Formatting_Views source path specified

INSERT INTO StartingFolderPipe VALUES (groupId,1,'LV','A',FV_FOLDER,LV_FOLDER);

-- Generate Business_Views from the Logical_Views source path specified

INSERT INTO StartingFolderPipe VALUES (groupId,1,'BV','A',LV_FOLDER,BV_FOLDER);

-- Generate Client_Views from the Business_Views source path specified

--　Path introspection is recursive starting at a base folder.

--　When you want to generate views from multiple folders in a hierarchy, provide the least common denominator folder as the base path and set the path derivation mode='D' for derived which will traverse all sub-folders during generation.

--　When using 'D', Consider using base paths for source and target so directories are emulated exactly.

INSERT INTO StartingFolderPipe VALUES (groupId,1,'CV','D',BV_FOLDER,CV_FOLDER);

-- Generate Client_Published from the Client_Views source path specified

--　Path introspection is recursive starting at a base folder.

--　When you want to generate views from multiple folders in a hierarchy, provide the least common denominator folder as the base path and set the path derivation mode='D' for derived which will traverse all sub-folders during generation.

--　When using 'D', Consider using base paths for source and target so directories are emulated exactly.

INSERT INTO StartingFolderPipe VALUES (groupId,1,'CP','D',CV_FOLDER,CP_FOLDER);

-- Generate CRUD operations from the CRUD source folder by designating it as the target folder.

-- By generating from the Physical_Views it will insure that no new or derived columns are propagated.

-- It is not permitted to perform a CRUD operation against a view with any derived columns present.

--DEPRECATED: Generate Physical_Views from the Physical_Metadata source path specified

INSERT INTO StartingFolderPipe VALUES (groupId,1,'PV','A',PM_FOLDER,PV_FOLDER);

SET CR_FOLDER=PV_FOLDER;

INSERT INTO StartingFolderPipe VALUES (groupId,1,'CR','A',CR_FOLDER,crudPath);

4. There are no changes to be made. Close the procedure.

6 Generate Physical Layer Formatting Views

OBJECTIVE: Generate the views for the Formatting layer.

General Note about what generateFormattingViews does:

a. This step changes the physical names to the logical or canonical names by reading from the files found in the following folder: C:\CompositeSoftware\BestPractices\Common_Model_v3_file[1-4].csv

b. It then compares the physical container and column name that it introspected from the physical metadata folder with values in these files to determine a logical name.

General note about the generation scripts:

c. The parameter "generateViewsWrapper" is defaulted to print results to the console window when null or 1.   If you want to see the output as a cursor, change the value to 0. However, please note the following:

d. The cursor output window will stop displaying when it hits the cursor limit which is configured in Administration→Configuration→Studio→Data→Fetch Rows Size and Cursor Fetch Limit

e. Modify the Cursor Fetch Limit to an arbitrary number such as 500. Any modification affects all Composite users.

DIRECTIONS:

1. Modify **ConfigureParams** to setup debug output for all view generation procedures

    a. Open /shared/labs/lab##/_scripts/Configure/ConfigureParams

    b. Scroll down to the bottom of the page

    c. Set Debug Level 1=Y

    d. Set Debug Level 2=Y

    e. Set Debug Level 3=N

    f. Save and Close

2. Make sure that Step 5 was finished to completion

3. Expand the folder to /shared/labs/lab##/_scripts/Generate

4. Generate the Formatting Views

    a. Open **generateFormattingViews()** procedure

b. Click Execute

    i. generateViewsWrapper=0

    ii. overwrite=2

    iii. copyAnnotation=1

    iv. copyPrivilegeMode=1

    v. exactMatch=1

    vi. derivedFilterPath=**"customers, orders"**

       1. Use double quotes around the pair to signify that they both belong to the ds_orders1 groupId.

       2. If you have a pair of group ids such as groupIds=ds_orders1, ds_orders2, then you could create a paired derivedFilterPath= "customers,orders", "orders,orderdetails". The "customers,orders" is paired with ds_orders1 and "orders,orderdetails" is paired with ds_orders2.

       3. Notice that the filter in this case is all lower case. It is exactly the same as what the physical metadata name is. This allows you to filter on the source by only generating views that potentially changed and not the entire group of views defined by the group id.

    vii. sourceResource=Null is checked

    viii. generateToFolder=Null is checked

    ix. groupId=ds_orders1

       1. If left null, then all rows marked with "FV" in ConfigureStartingFolders will be used to target generating formatting views from the physical metadata.

       2. One or more group ids may be provided in a comma separated list.

    x. Scroll down to the DECLARE generateCast option and review:

       1. DECLARE generateCast   SMALLINT DEFAULT 2;

       2. Used when generateMode='G' or 'R'. This parameter allows the user to control whether to generate the cast statement around the generated column or not. It uses the column type from the source view. The default of 2 is set so that no CAST statements are placed around columns that contain indexes. The use of the CAST statement can prevent the CIS optimizer from utilizing the indexes

affectively for some databases. The best practice is to not generate the CAST in the formatting layer for columns. However, if the user is really trying to CAST the column to a different type for display purposes, they might decide that option 3 is better which will automatically generate a display column for the column determined to have an index while not putting any CAST statements around the index column.

3.   0=Do not generate CAST statement. Pass through column as is. Default behavior.

4.   1=Generate the CAST statement around the column

5.   2-Generate the CAST statement around the non-index columns only (No CAST on index columns)

6.   3-Generate the CAST statement around the non-index columns only and generate a "display" CAST column for each index column. (No CAST o nindex columns)

7.   4-Generate the CAST statement around the non-index columns and non-primary key index columns only (No CAST on primary key index columns)

8.   5-Generate the CAST statement around the non-index columns and non-primary key index columns only and generate a "display" CAST column for each primary key index column. (No CAST on primary key index columns)

xi.   Click OK

c. Refresh Studio

BACKGROUND ON PARAMETER OPTIONS:

generateViewsWrapper

- 0 – print the output to the cursor. The cursor is bound by Composite Studio "Fetch Row Size" and "Cursor Fetch Limit". The cursor stops producing output when it hits those limits. The limits are configured in Composite Studio Administration Configuration Studio Data Fetch Rows Size and Cursor Fetch Limit. Modify the Cursor Fetch Limit to an arbitrary number such as 500

- 1 (default) – do not print the cursor output to the cursor but redirect to the console window. The aforementioned limits do not apply.

overwrite – allows user to decide whether they want to overwrite an existing view or not.

- 0="FAIL_IF_EXISTS" – do not overwrite the resource. If the resource exists, raise an exception.

- 1="SKIP_IF_EXISTS" – skip the resource if it exists and continue processing

- 2 (default)="OVERWRITE_IF_EXISTS" – do overwrite the resource if it exists.

copyAnnotation – allows user to decide whether they want to copy annotations or not form both resource and columns.

- 0 (default)=false – do not copy the annotation from the target resource

- 1=true – do copy the annotation from the target resource

copyPrivilegeMode – flag indicating the mode in which to copy privileges. Privileges are only copied from the parent when creating new resources including folders.

- null (default) – do not set any privileges at all

- 0 – set mode to "OVERWRITE_APPEND" - merges and does not update privileges for users or groups not mentioned.

- 1 – set the mode to "SET_EXACTLY" - makes privileges look exactly like those provided in the call.

exactMatch – specifies how the source resource will be matched against the resource being interrogated.

- 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath

- 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath

DIRECTIONS:

5. Review the folder: /shared/labs/lab##/Physical/Formatting

    a. ds_orders1

        i. Customers

        ii. Orders

6. Compare the Spreadsheet with the "Customers" generated view

    a. Open Common_Model_v3_file4_sample_lab.xlsx (D: or C:/CompositeSoftware/BestPractices/BestPractices_vXX)

    b. Search for your lab number (lab##).

c. Compare the /Formatting/ds_orders1/Customers view with the lines 3-17 (ds_orders1.customers) in the spreadsheet.

d. Notice the use of Logical Type and Logical Transformation. Now look at the Customers view for the transformations from the spreadsheet.

e. Notice line 17 and how there is no Physical Name.   This is how to create a new or derived column. In this case both the Logical Type and Logical Transformation are required.

f. Notice on line 15 that a "?" is used in place of the physical column name. The generation scripts will replace the column name with the actual name. This is done so that it makes it easier to copy and paste logical transformations from one line to the next. The example is: CASE ? WHEN '' THEN NULL ELSE ? END

g. Note: It is sometimes easier to prototype a complex logical transformation in CIS first and then paste the results into this column. Many times, once you have established a pattern of transformations, it is easy to copy and paste from spreadsheet line to spreadsheet line.

7. Generate the Formatting Views

   a. Open **generateFormattingViews()**

      i. In this portion of the lab, you will experiment with different combinations of more ore more group ids in a comma separated list.

   b. Try groupId=ds_orders1

      i. Click Execute

      ii. generateViewsWrapper=1

         1. No cursor output. Only output to console window.

      iii. derivedFilterPath=

         1. Click the "Null" checkbox to the right of the entry box

      iv. Refresh Studio …What got generated?

   c. Try groupId= ds_inventory, ds_XML

      i. Click Execute

      ii. generateViewsWrapper=0

         1. Output results to cursor window.

      iii. derivedFilterPath=

      iv. Refresh Studio …What got generated?

   d. Try groupId=testfile, Common_Model_v2

    i.   Click Execute

    ii.   generateViewsWrapper=1

    iii.   derivedFilterPath=

    iv.   Refresh Studio …What got generated?

7 Generate Business Layer Logical Views

DIRECTIONS:

1. Generate the Business Layer Logical Views

   a. Open **generateLogicalViews()**

   b. Click Execute

       i.   generateViewsWrapper=0

       ii.   derivedFilterPath=

       iii.   groupId=ds_orders1

   c. Refresh Studio

   d. Review the folder: /shared/labs/lab##/Business/Logical

   e. Note: The same concept of "groupId" can be used when generating logical views.

8 Generate Business Layer Business Views

DIRECTIONS:

1. Generate the Business Layer Business Views

   a. Open **generateBusinessViews()**

   b. Click Execute

       i.   generateViewsWrapper=0

       ii.   derivedFilterPath=

       iii.   groupId=ds_orders1

   c. Refresh Studio

   d. Review the folder: /shared/labs/lab##/Business/Business

   e. Note: The same concept of "groupId" can be used when generating logical views.

9 Generate Application Layer Views

DIRECTIONS:

1. Generate the Application Views (Client Views)

a. Open **generateClientViews()**

b. Click Execute

    i. generateViewsWrapper=0

    ii. derivedFilterPath=

    iii. groupId=ds_orders1

c. Refresh Studio

d. Review the folder: /shared/labs/lab##/Application/Views (Client Views)

e. Note: The same concept of "groupId" can be used when generating client views.

## 10 Generate Application Layer Published Views

DIRECTIONS:

1. Generate the Client Published

a. Open **generateClientPublished()**

b. Click Execute

    i. generateViewsWrapper=0

    ii. derivedFilterPath=

    iii. groupId=ds_orders1

c. Refresh Studio

d. Review the folder: /shared/labs/lab##/Application/Published (Client Published)

e. Note: The same concept of "groupId" can be used when generating client views.

11 Generate Cast Views

DIRECTIONS:

1. Create a new view in the /Business/Business folder called "**Orders_Open**"

a. Drag /Business/Logical/ds_orders1/Orders into it

b. Drag /Business/Logical/ds_orders1/Order_Details

c. Join them together on Order_Id by

    i. Drag Order_Id from Orders to Order_Id in Order_Details
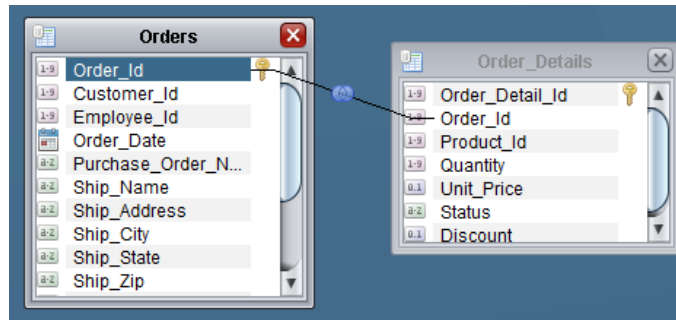
d. Go to the "Grid" view and add the columns for both tables using the List
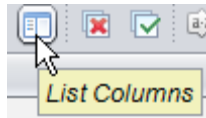
Column:
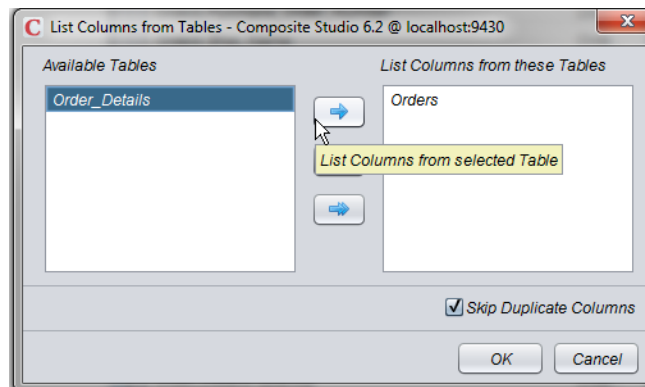
i. Select the 2nd instance of the column Order_Details.Order_Id found towards the latter 1/3 of the screen as it is a duplicate and remove it

e. Add a where clause from the Grid view by clicking in the Criteria

    Criteria column for the "Status" field and type "= 'open' " and save

i. Order_Details.Status = 'open'

Expanded Picture:

Column	Alias	Table	Output	Sort Type	Sort Order	Group By	Criteria	Or ...
Orders.Order_Id		Orders	☑					
Orders.Customer_Id		Orders	☑					
Orders.Employee_Id		Orders	☑					
Orders.Order_Date		Orders	☑					
Orders.Purchase_Order_Number		Orders	☑					
Orders.Ship_Name		Orders	☑					
Orders.Ship_Address		Orders	☑					
Orders.Ship_City		Orders	☑					
Orders.Ship_State		Orders	☑					
Orders.Ship_Zip		Orders	☑					
Orders.Ship_Country		Orders	☑					
Orders.Ship_Phone_Number		Orders	☑					
Orders.Ship_Date		Orders	☑					
Orders.Shipping_Method_Id		Orders	☑					
Orders.Freight_Charge		Orders	☑					
Orders.Data_Source		Orders	☑					
Order_Details.Order_Detail_Id		Order_Deta...	☑					
Order_Details.Product_Id		Order_Deta...	☑					
Order_Details.Quantity		Order_Deta...	☑					
Order_Details.Unit_Price		Order_Deta...	☑					
Order_Details.Status		Order_Deta...	☑				= 'open'	
Order_Details.Discount		Order_Deta...	☑					

    f. Add Annotation:

        i. Click on the "Info" tab and add the following annotation:

            This view queries open orders only.

    g. Execute the view to insure it works and returns only 'open' orders

    h. Leave the view open for the next step

2. Generate a Cast View

    a. Copy the resource path from the "Info" tab of the Open_Orders view

    b. Open **generateCastViews()**

    c. Click Execute

        i. generateViewsWrapper=0

        ii. overwrite=2

        iii. copyAnnotation=1

        iv. copyPrivilegeMode=1

        v. exactMatch=1

        vi. sourceResource= /shared/labs/lab##/Business/Business/Orders_Open

            1. replace ## with your lab id…e.g. lab00

        vii. targetResource= /shared/labs/lab##/Application/Published

            1. replace ## with your lab id…e.g. lab00

    d. Click OK

    e. Refresh Studio

  f. Review the /Application/Published folder to find "Orders_Open"

    i. Open it and observe the cast statements. The Published folder provides a contract with client applications. There should be no logic in these views. There are only cast statements which serve as an insulator of change between the application and views below the Published sub-layer. The reason this is important is that there are BI applications that will introspect Composite to bring in the Composite metadata and if the metadata changes, it will cause unexpected runtime behavior by the BI application.

    ii. Review the Annotation on the "Info" tab.

12 Generate Published Database Views

  DIRECTIONS:

  1. Create a **lab##** database
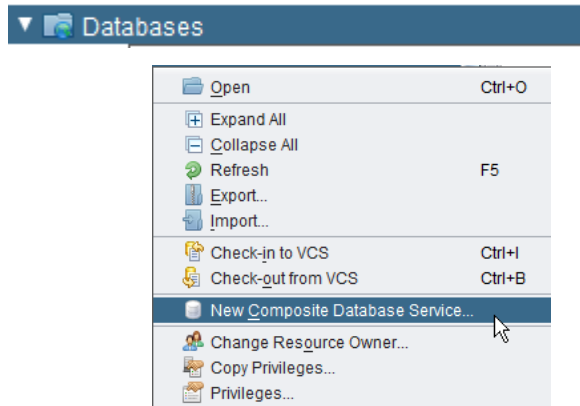
    a. Result:

   i. Example lab00 database: lab00

  2. Generate the Published Database

    a. Open **generatePublishedResource()**

    b. Click Execute

      i. generateViewsWrapper=0

      ii. overwrite=2

      iii. copyAnnotation=1

      iv. copyPrivilegeMode=1

      v. sourceResource= /shared/labs/lab##/Application/Published

      vi. generateToFolder= /services/databases/lab##

      vii. derivedFilterPath=

        viii.   groupId=

   c. Refresh Studio

   d. Review the "Published Database" folder:

        i.   /Composite Data Services/Databases/lab##

   e. Open "Orders_Open"

        i.   Review the Annotation on the "Info" tab. The annotation indirectly came from the Open_Orders view that you created in step 11.

        ii.   Execute the table to see the open orders.

   f. Note: The same concept of "groupId" can be used when generating published resources.

13 Generate Data Abstraction Spreadsheet

DIRECTIONS:

1. Generate the Data Source List CSV

   a. Open **generateDatasourceListCSV**

   b. Click Execute

        i.   csvFullPath= /temp/lab##_Common_Model_v2_file.csv

            1. replace ## with your lab id…e.g. lab00

        ii.   bufferSize=100

        iii.   generateHeader=0

        iv.   generateLogicalNames=1

        v.   generateMode=R

        vi.   caseRule=

        vii.   useAliasRule=

        viii.   generateUnsupportedColumnType=

        ix.   exactMatch=

        x.   derivedFilterPath=

        xi.   targetResource=

        xii.   layerType=FV

        xiii.   groupIds=ds_orders1

   c. Using Windows Explorer, look for the file in the /temp directory either on C: or D: drive

   d. Open the file with Excel

    i. Columns A-I were generated.

    ii. **Round-Trip Synchronization**:

A user may now copy columns A-I starting at row 2 through the end and paste them back into the original Common_Model_v3_file4_sample_lab.xlsx spreadsheet.

This will allow the user to keep CIS synchronized with the spreadsheet.

14 Search for a resource

DIRECTIONS:

1. Search for a resource anywhere in the starting folder that you provide.

    a. Open /Display/**searchResourceTree()**

    b. Click Execute

        i. resourcePath= /shared/labs/lab##

        ii. resourceName= Orders_Open

        iii. ignoreCase=Y

    c. Result:

ResourceName	ResourcePath	ResourceType	ResourceSubType
Orders_Open	/shared/labs/lab##/Application/Published	TABLE	SQL_TABLE
Orders_Open	/shared/labs/lab##/Business/Business	TABLE	SQL_TABLE

15 Generate CRUD Views (Create, Read, Update, Delete)

1. **Skip this section if it is not applicable**

DIRECTIONS:

2. Generate the Physical Procedures

    a. Open **generatePhysicalViews()**

        i. Even though this procedure is deprecated, it may still be used to generate views that do not have any new or derived columns. When performing CRUD operations, the views may not contain any new or derived columns. The closer to the physical source the better off the user will be when executing CRUD operations.

    b. Click Execute

        i. generateViewsWrapper=0

        ii.   derivedFilterPath=

       iii.   groupId=ds_orders1

   c. Refresh Studio

   d. Review the folder: /shared/labs/lab##/Physical/Physical/ds_orders1

3. Generate the CRUD Procedures

   a. *Note*: This operation is only necessary if you are performing Create, Update, or Delete operations on the generated views. Transactions are only supported for a single database source at this time.

   b. Open the /Application/Services/CRUD folder – there should not be anything there. If there is, then delete it.

   c. Open **generateCRUDOperations()**

   d. Click Execute

       i.   generateViewsWrapper=0

       ii.   layerType=CR

       iii.   procedureName: leave null (it will generate the name and the groupId appended).

       iv.   groupId=ds_orders1

   e. Refresh Studio

   f. Review the folder: /shared/labs/lab##/Application/Services/CRUD

       i.   Review **/Definitions** – A "TypeDefinitionsGen_ds_orders1" procedure was created.

           1. It contains "DECLARE PUBLIC TYPE <table_Type> ROW ()" definitions for each table.   These public types are used across all of the CRUD procedures that were generated.

       ii.   Review **/Coordinate** – the coordinate functions provide a central access method for executing on the CRUD operation. It can be exposed as a web service.

       iii.   Review **/Create** – this provides the "Create" operation (insert).

       iv.   Review **/Read** – this provides the "Read" operation (select by pk).

       v.   Review **/Update** – this provides the "Update" operation (update).

       vi.   Review **/Delete** – this provides the "Delete" operation (delete).

       vii.   Review **/RetrievePK** – this provides a way to select data by primary key.

viii. Review **/isEmpty** – this provides a way to test whether a record is empty or not.

g. Open **/CRUD/Read/get_customers_ById**

i. Execute and enter 3 for the CustomerID and nothing for the LogIdentifier

ii. Row # 3 is returned from the ds_orders1.customer table.

5   Summary

Congratulations.

In this lab, you have had an opportunity to execute all of the Data Abstraction Best Practices generation scripts.   The key to these scripts is setting up a proper ConfigureStartingFolders() procedure. The best approach is to set up sub-folders inside /Physical/Metadata for each data source. Set up a section in ConfigureStartingFolders() for each of those sub-folders. If a data source such as an Oracle database has multiple schemas that are needed, it is recommended to set up a section for each schema. This gives you a finer-grained level of control over what you can control during the generation.