How To Use Data Abstraction Best Practices View Generation

An Open Source Asset for use with TIBCO® Data Virtualization

TIBCO Software empowers executives, developers, and business users with Fast Data solutions that make the right data available in real time for faster answers, better decisions, and smarter action. Over the past 15 years, thousands of businesses across the globe have relied on TIBCO technology to integrate their applications and ecosystems, analyze their data, and create real-time solutions. Learn how TIBCO turns data—big or small—into differentiation at www.tibco.com.

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

www.tibco.com

Global Headquarters
3303 Hillview Avenue
Palo Alto, CA 94304

Tel: +1 650-846-1000
+1 800-420-8450
Fax: +1 650-846-1005

Revision History

Version	Date	Author	Comments
1.0	04/13/2010	Mike Tinius	Initial revision
2.0	07/20/2010	Mike Tinius	
3.0	08/25/2010	Mike Tinius	Added Create, Read, Update and Delete (CRUD)
4.0	02/08/2011	Mike Tinius	Improved instructions. Upgraded utilities.
5.0	11/05/2011	Mike Tinius	Simplified folder structure. Modified Excel files. Upgraded utilities.
5.1	02/15/2012	Mike Tinius	Allow for multiple schemas per data source. Modified Excel files. Upgraded utilities.
6.0	05/14/2012	Mike Tinius	Added new features. Upgraded utilities.
6.1	07/30/2012	Mike Tinius	Added controls for generating indexes.
6.2	8/6/2012	Mike Tinius	Fixed bugs
6.3	9/27/2012	Mike Tinius	Fixed issue where spaces/underscores preceding/following were trimmed.
6.4	10/01/2012	Mike Tinius	Added generateCastViews
6.5	10/30/2012	Mike Tinius	Added support for Utilties_2012Q4.car and Utilities_2012Q4_61+.car
6.6	11/15/2012	Mike Tinius	Added support for Utilties_2012Q401.car and Utilities_2012Q401_61+.car and documentation generation.
7.0	04/19/2013	Mike Tinius	Added generateDatasourceListCSV, synchronized with Utilities_2013Q104, change Common_Model_v2_file[1-3].csv spread sheet format and modified Best Practices from 4 major layers to 3.
7.1	06/04/2013	Mike Tinius	Fixed bugs found in 7.0. Added generatePublishedResource and upgradeProject procedures. Added parameters: overwrite, copyAnnotation, and copyPrivileges to all generate procedures
7.2	06/28/2013	Mike Tinius	Provided the ability to install Best Practices 7.2 in parallel with earlier releases to make migration easier.
7.3	08/30/2013	Mike Tinius	Synchronized with Utilities_2013Q301.car. Moved BestPractices_vXX folder to /shared/Utilities. Enhanced upgradeProject. Fixed header printout for generateDatasourceListCSV. Fixed performance issue with generateViews. Fixed an issue with CRUD procedure generation.
8.0	12/05/2013	Mike Tinius	Repackaged as /shared/PSAssets/BestPractices_8_0. Modified spreadsheet format.
8.1	02/21/2014	Mike Tinius	Updated with Utilities_2014Q1.car.
8.11	04/01/2014	Mike Tinius	Fixes for 8.1.1 Fixed issues with generateDatasourceList. Resolved upgrade from

			8.0 to 8.1 and 8.1.1.
8.12	04/28/2014	Mike Tinius	Rebranding PS Assets as AS Assets. Requires Utilities_2014Q2.car.
8.13	08/08/2014	Mike Tinius	Updated generateViews methods with new generateCast variable features. Updated to use Utilities_2014Q3.car
8.14	08/25/2014	Mike Tinius	Updated to use Utilities_2014Q301.car and modified upgrade capabilities.
8.15	11/26/2014	Mike Tinius	Updated to use Utilities_2014Q4 + several other enhancements and fixes.
8.16	05/20/2015	Mike Tinius	Updated for Best Practices v8.1.6 – Powerpoint format only.
8.17	09/21/2015	Mike Tinius	Updated for Best Practices v8.1.7 – added generateViews=2 to allow generating views with a SELECT * projection. Requires Utilities_2015Q3.car
8.172	12/11/2015	Mike Tinius	Fixed "generateViews" to allow a derived filter path of LONGVARCHAR.
8.18	05/24/2017	Mike Tinius	Updated for Best Practices v8.1.8 – added Privilege scripts and manage annotations.
8.19	12/06/2017	Mike Tinius	Transitioned to Tibco for release 8.1.9
2018Q1	03/20/2018	Mike Tinius	Release 2018Q1 – updated to use Utilities 2018Q1. Added major capability: Dynamic File Framework. Changed references of mysql to postgres.
2019Q1	01/25/2019	Mike Tinius	Release 2019Q1 - Added the ability to handle resourceCaseRule, columnCaseRule, resourcePrefix, resourceSuffix and newColumnList for generateMode='G'.
2019Q101	01/29/2019	Mike Tinius	Release 2019Q101 – Fixed bug so derivedFilterPath could be used with generateToFolder.
2019Q200	06/13/2019	Mike Tinius	Release 2019Q200: 1. Added scriptsPath for flexibility of the location of the _scripts folder. 2. Fixed a bug in generateProject. 3. Removed all references to custom functions in favor of calling explicit paths so as to avoid name collisions at customer sites. 4. Converted as many vectors to XML as possible: ConfigParamVector and startingFolderVector. 5. Retained the original interfaces that contain vectors for backwards compatibility but the implementation procedures will use XML as input instead of vectors. 6. Fix a bug with derivedFilterPath correlating with

			the list of groupIds.
2019.300	08/01/2019	Mike Tinius	Modified upgradeProject and generatProject. Fixed bug in generateDatasourceListXML(). Set minimum utilities to 2019.301. Modified all occurrences of getConstant to getConstantV2. Modified Documentation triggers, constants and added documentationDriverWrapper. Modified generateDatasourceListXMLInsertDB.
2020.200	03/12/2020	Mike Tinius	Requires Utilities 2020Q200. Fixed bug when generating a view for a PROCEDURE and generateViews=2. Changes in various procedures due to Utility changes in getBasicResourceCursor_SQL_TABLE and getBasicResourceCursor_SQL_PROCEDURE_CURSOR. Code changes DO NOT affect existing project resources.
2020.201	05/12/2020	Mike Tinius	Bug fix for pqCreate_postgres_cache_tables. Dynamic File Framework changes.

Related Documents

Name	Version
How To Use Utilities.pdf	2020Q200
Data Abstraction Best Practices Installation and Release Notes	2020Q201
How To Use Data Abstraction Best Practices View Generation.pdf	2020Q201
How To Test Data Abstraction Best Practices View Generation.pdf	2020Q200
How To Learn Data Abstraction Best Practices View Generation.pdf	2020Q200
How To Use Data Abstraction Best Practices Manage Annotations.pdf	2020Q200
How To Use Data Abstraction Best Practices Privilege Scripts.pdf	2020Q200
How To Use Data Abstraction Best Practices Dynamic File Framework.pdf	2020Q201

Supported Versions

Name	Version
TIBCO® Data Virtualization	7.0 or later
AS Assets Utilities open source	2020Q200 or later

Table of Contents

1 Introduction

Purpose

The purpose of Best Practices generation scripts is to generate the different layers of the Data Abstraction Best Practices.

In Best Practices v7.0 and higher, the layers have changed and thus the generation scripts have changed. The following provides a diagram of what the layers currently look like:

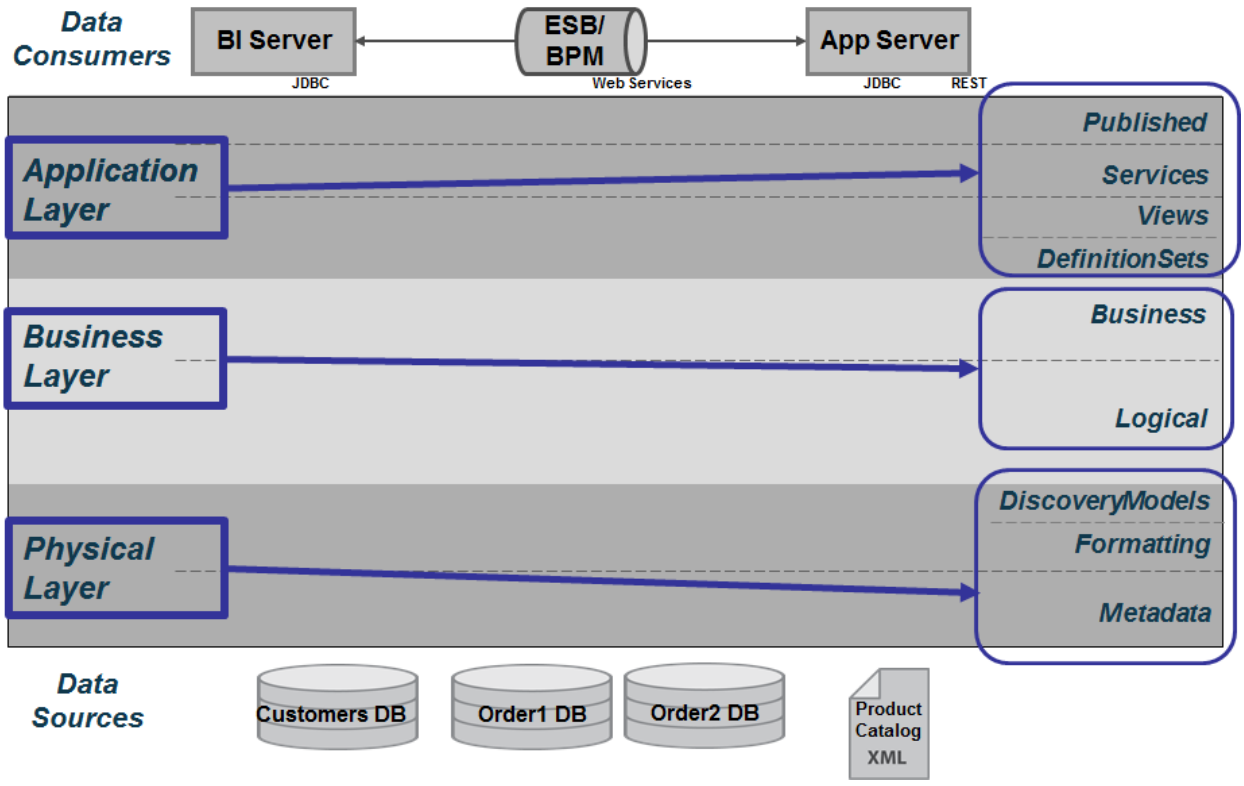


Figure one: Technical Data Abstraction Layers

The main change is that v7.0 goes from 4 major layers to 3. The sub-layer changes include removing Physical Views, sliding Formatting Views down into the Physical layer and removing Federated Views.

This document provides information on these various topics:

1. **Physical Metadata Layer** – This layer is imported using the Composite Studio "New Data Source" wizard.
2. **Generate Physical Views** – This sub-layer and the generation procedures associated with it have been deprecated. The "generatePhysicalViews()" and is still available for

backwards compatibility if needed. However, it may be useful to use Physical Views when generating the CRUD operations.

3. **Generate Formatting Views** – Generate Formatting Views from Physical Metadata utilizing the spreadsheets and the procedure "generateFormattingViews()".

4. **Generate Logical Views** – Generate Logical Views from Formatting Views utilizing the procedure “generateLogicalViews()”. If the views are derived directly from the Formatting Views with no joins, then it is possible to generate them. If joins are required then these views will have to be built by hand.

5. **Generate Application Views** – Generate Application Views from Logical Views or Business Views utilizing the procedure “generateClientViews()”. Similar to Logical Views, it is possible to generate these views from the Logical as long as there are no joins required. It is possible to generate them using a Logical Model to Client Object Model spreadsheet if needed or simply generate from the same names as the Logical names.

6. **Generate Application Published Views** – Generate Application Published from Application Views utilizing the procedure “generateClientPublished()”. The objective of the client published views is to provide casting of any view in the lower layers including Application Views, Business Views, or Logical Views.

7. **Generate Published Resources** – Generate “links” from the views in the Application layer to a Composite database. These are published resources that are accessible via JDBC, ODBC and OData.

8. **Generate CRUD Operations** – Generate Create, Read, Update, and Delete (CRUD) operations for use when performing Read/Write against a database repository. You would use the generate “generateCRUDOperations()” method to achieve this.

9. **Generate Type Definitions** – Generate type definitions provide a common set of public type definitions that are used with the CRUD procedures or may be used with other custom procedures. They provide a centralized location for all procedures to use. You would use the generate “generateTypeDefinitions()” method to achieve this.

10. **Generate Casting Views** – This provides the ability to generate views at any layer with casting by providing a source and target resource using the “generateCastViews()” procedure.

11. **Generate Data Source List** – This provides the ability to generate the list of resources in the Formatting Views or Physical Metadata to a cursor output or CSV file. Use the methods the following methods to accomplish this task: “generateDatasourceList()”, “generateDatasourceListCSV()” and “generateDatasourceListInsertDB”.

Audience

This document is intended to provide guidance for the following users:

1. **Architects** – Architects will achieve a deeper level of understanding of how the Best Practices scripts can be utilized from a project perspective.
2. **Developers** – Developers will learn how to use the scripts to generate views.

References

Product references are shown below. Any references to CIS or DV refer to the current TIBCO® Data Virtualization.

- TIBCO® Data Virtualization was formerly known as
  - Cisco Data Virtualization (DV)
  - Composite Information Server (CIS)

2 Helpful Tips

Tips for using the Best Practices scripts

This section discusses helpful tips when using the Data Abstraction Best Practices scripts:

Setup Caching for Common Model Spreadsheets to a Database

1. Relational Database Cache (recommended)

1.1. Modify Spreadsheet Data Sources

1.1.1. Open Composite Studio to modify location of spreadsheet files

1. Modify CommonModelExcelSources data source path settings

   /shared/ASAssets/BestPractices_v81/DataSource/CommonModelExcelSources

   Default Root Path: C:/DV/BestPractices/BestPractices\BestPractices_v80

2. Modify CommonModelCSVSources data source path settings [optional]

   /shared/ASAssets/BestPractices_v81/DataSource/CommonModelExcelSources

   Default Root Path: C:/DV/BestPractices/BestPractices\BestPractices_v80

3. Save both resources

4. **NOTE: Only one source will be used at a time based on /shared/ASAssets/BestPractices_v81/_ProjectMaintenance/defaultValues. CommonModelType [EXCEL, CSV, DB] – default=EXCEL**

   1.1.1.4.1. EXCEL – uses CommonModelExcelSources

   1.1.1.4.2. CSV – uses CommonModelCSVSources

   1.1.1.4.3. DB – used by generateDatasourceListInsertDB to generate physical to logical mappings and store them into the common_model_v3 table. No spreadsheet is referenced with this setting.

1.2. Option 1 [recommended] – Postgres Cache Database

Setup Steps

1.2.1. Modify CommonModelCache data source connection settings using Studio

/shared/ASAssets/BestPractices_v81/DataSource/CommonModelCache

Note: Use equivalent settings for your environment for the connection information.

- Host: localhost
- Port: 9408 (whatever your postgres port is 9400+8)
- Database Name: ciscache (the default cache database)
- Login: root (default)
- Password: (whatever your postgres root password is)

1. Save and Test the connection.
2. Execute the script: /shared/ASAssets/BestPractices_v81/DataSource/CacheInstructions/pqCreate_postgres_cache_tables
3. Reintrospect CommonModelCache

1.2.2. Refresh common_model view

1. Enable caching and save

   /shared/ASAssets/BestPractices_v81/DataSource/Physical/common_model

2. Execute a cache refresh on common_model view

   Note: This may take a few minutes depending on the number of rows in the three spreadsheets…Common_Model_v3_file[1-4].xlsx.

3. Monitor progress in the Manager, Cached Resources window

1.3. Option 2 [not recommended] – set up your own cache using your own relational database.

1.3.1. While this is possible, it is not recommended. It will be easier just to use the built in postgres cache database from step 1,1 above. There is much more setup with this option.

1.3.2. Create a new relational data source

1. Configure all of the correct parameters to connect to your database.
2. Click the “Test Connection” to verify that you can connect. Proceed to 1.2.3 below.

1.3.3. Create tables:

1. Locate the SQL from the following package query and convert the SQL to your chosen database SQL and execute the converted SQL:

/shared/ASAssets/BestPractices_v81/DataSource/CacheInstructions/pqCreate_postgres_cache_tables

2. Introspect [add/remove] your new data source and bring in the following tables:

   cache_status, cache_tracking, common_model, and common_model_v3

3. Rebind the following view to the database table common_model_v3: /shared/ASAssets/BestPractices_v81/DataSource/common_model_v3

1.3.4. Click “Refresh” to refresh the cache.

1. Monitor progress in the Manager, Cached Resources window

1.4. Potential Errors when caching

1.4.1. The following error is vague but indicates an underlying problem with the caching.

CANCELLED. Cause:
com.compositesw.common.workflow.Cancellable$CancellationException: Request was terminated. [SELECT datasourceName,projectFolderName,greatGrandParentName,grandParentName,parentName,containerName,resourceName,resourceNum,logicalName,l...

If you see the above error, verify that the spreadsheet contains a “**logicalType**” value for any “**resourceNames**” that are blank. It is an error for this scenario to be true. Look in the “**System Message**” column of the spreadsheet for this error or execute “/shared/ASAssets/BestPractices_v81/DataSource**/pCommon_Model_Union**” and retrieve all of the rows in order to validate that the following error is not being raised:

com.compositesw.cdms.webapi.WebapiException: AnonymousProcedure.ex: SPREADSHEET ERROR (Common_Model_v3_file[1-3].xlsx): Logical Type required. New fields that are not inherited from a physical view must be assigned a logical type.

Resolution: Provide a “**logicalType**” value for any “**resourceNames**” that are blank and recache the common_model view.

Typical parameter settings

2. Typical parameter settings

2.1. Parameter: “**overwrite**” – typically this is set to 2 which is the default because you would want to overwrite any existing views when you are re-creating them

2.2. Parameter: “**copyAnnotation**” – the default is 0 (false) to not copy annotations, but since generating views is 1-to-1, it may make more sense to set this to 1 to copy the annotations.

2.3. Parameter: “**copyPrivilegeMode**” – the default is not to copy privileges at all, but if your environment makes use of privileges, it would make more sense to set this to 1.

Configure Starting Folders Concepts

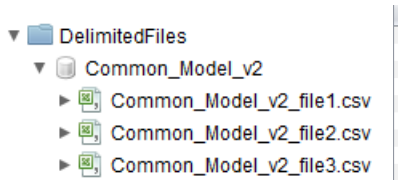
3. ConfigureStartingFolders Concepts

3.1. The “ConfigureStartingFolders” is a procedure that provides the resource path into the different Data Abstraction Best Practices layers and is organized around each data source and transformation folder. There are code designations for each layer type. There is a unique groupId that provides knowledge for each data source grouping. The layer type and group id are a couple of the parameters used by the generation scripts. Some of the generation scripts such as generateFormattingViews() will embed the layerType=FV by default whereas the more generic generatViews() allows the user to designate the layer type.

3.2. ***Group Id***: The groupId is automatically generated for each data source based on the data source name, catalog name and schema name. For the /Formatting/Transformations folder the name is based on the folders that occur under the /Transformations folder. The best approach for XML transformations is to group them in folders that mirror the names used in the data source. However the limitation is that a folder should not contain a hierarchy of transformations. Keep the folder groupings flat or there will be duplicates generated.

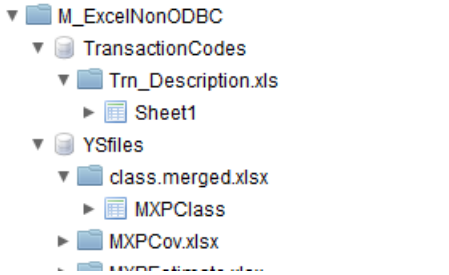
3.2.1. GROUPID DELIMITED FILES

1. The /Metadata/DelimitedFiles/Common_Model_v2 data source contains three CSV files. These are a single gouping which which is represented in the picture on the right as “Common_Model_v2”. The groupId is generated from the data source name only because there are no subordinate containers.



groupId	process	layerType
Common_Model_v2	1	DB
Common_Model_v2	1	PM
Common_Model_v2	1	FV
Common_Model_v2	1	LV
Common_Model_v2	1	BV
Common_Model_v2	1	CV
Common_Model_v2	1	CP
Common_Model_v2	1	PV
Common_Model_v2	1	CR

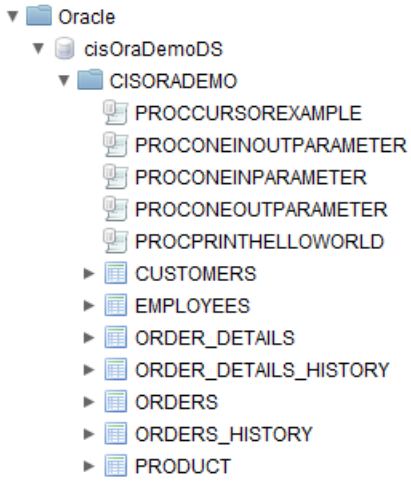
2. The /Metadata/M_ExcelNonODBC/TransactionCodes data source contains one Excel file with one sheet. These are a single gouping which which is represented in the picture on the right as “TransactoinCodes.Tm_Description.xls”. The groupId is generated from the data source name and the Excel file name.



groupId	process	layerType
TransactionCodes.Trn_Description.xls	1	DB
TransactionCodes.Trn_Description.xls	1	PM
TransactionCodes.Trn_Description.xls	1	FV
TransactionCodes.Trn_Description.xls	1	LV
TransactionCodes.Trn_Description.xls	1	BV
TransactionCodes.Trn_Description.xls	1	CV
TransactionCodes.Trn_Description.xls	1	CP
TransactionCodes.Trn_Description.xls	1	PV
TransactionCodes.Trn_Description.xls	1	CR

3.2.2. GROUPID SCHEMA EXAMPLE

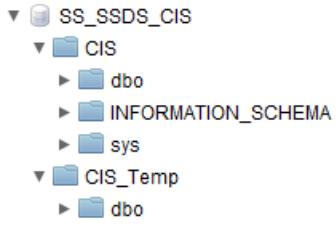
1. The /Metadata/Oracle/cisOraDemoDS data source only contains schemas. The data source and schema is used to generate the groupId.



groupId	process	layerType
cisOraDemoDS.CISORADEMO	1	DB
cisOraDemoDS.CISORADEMO	1	PM
cisOraDemoDS.CISORADEMO	1	FV
cisOraDemoDS.CISORADEMO	1	LV
cisOraDemoDS.CISORADEMO	1	BV
cisOraDemoDS.CISORADEMO	1	CV
cisOraDemoDS.CISORADEMO	1	CP
cisOraDemoDS.CISORADEMO	1	PV
cisOraDemoDS.CISORADEMO	1	CR
ds_hr.HR	1	DB
ds_hr.HR	1	PM
ds_hr.HR	1	FV
ds_hr.HR	1	LV
ds_hr.HR	1	BV
ds_hr.HR	1	CV
ds_hr.HR	1	CP
ds_hr.HR	1	PV

3.2.3. GROUPID CATALOG EXAMPLE

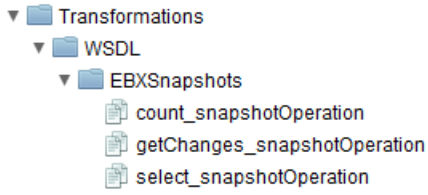
1. The /Metadata/SQL_Server/SS_SSDS_CIS data source contains a catalog and schema. Each catalog and schema combination results in a unique groupId being generated. The data source below contains two catalogs. The first catalog “CIS” contains three schemas: dbo, INFORMATION_SCHEMA, sys. The second catalog “CIS_Temp” contains one schem “dbo”.



groupId	process	layerType
SS_SSDS_CIS.CIS.INFORMATION_SCHEMA	1	PM
SS_SSDS_CIS.CIS.INFORMATION_SCHEMA	1	FV
SS_SSDS_CIS.CIS.INFORMATION_SCHEMA	1	LV
SS_SSDS_CIS.CIS.INFORMATION_SCHEMA	1	BV
SS_SSDS_CIS.CIS.INFORMATION_SCHEMA	1	CV
SS_SSDS_CIS.CIS.INFORMATION_SCHEMA	1	CP
SS_SSDS_CIS.CIS.INFORMATION_SCHEMA	1	PV
SS_SSDS_CIS.CIS.INFORMATION_SCHEMA	1	CR
SS_SSDS_CIS.CIS.dbo	1	DB
SS_SSDS_CIS.CIS.dbo	1	PM
SS_SSDS_CIS.CIS.dbo	1	FV
SS_SSDS_CIS.CIS.dbo	1	LV
SS_SSDS_CIS.CIS.dbo	1	BV
SS_SSDS_CIS.CIS.dbo	1	CV
SS_SSDS_CIS.CIS.dbo	1	CP
SS_SSDS_CIS.CIS.dbo	1	PV
SS_SSDS_CIS.CIS.dbo	1	CR
SS_SSDS_CIS.CIS.sys	1	DB
SS_SSDS_CIS.CIS.sys	1	PM
SS_SSDS_CIS.CIS.sys	1	FV
SS_SSDS_CIS.CIS.sys	1	LV
SS_SSDS_CIS.CIS.sys	1	BV
SS_SSDS_CIS.CIS.sys	1	CV
SS_SSDS_CIS.CIS.sys	1	CP
SS_SSDS_CIS.CIS.sys	1	PV
SS_SSDS_CIS.CIS.sys	1	CR
SS_SSDS_CIS.CIS_Temp.dbo	1	DB
SS_SSDS_CIS.CIS_Temp.dbo	1	PM
SS_SSDS_CIS.CIS_Temp.dbo	1	FV
SS_SSDS_CIS.CIS_Temp.dbo	1	LV
SS_SSDS_CIS.CIS_Temp.dbo	1	BV
SS_SSDS_CIS.CIS_Temp.dbo	1	CV
SS_SSDS_CIS.CIS_Temp.dbo	1	CP
SS_SSDS_CIS.CIS_Temp.dbo	1	PV
SS_SSDS_CIS.CIS_Temp.dbo	1	CR

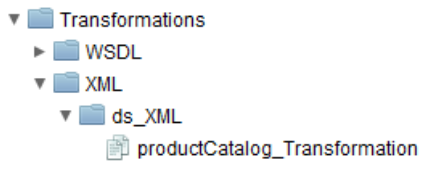
3.2.4. GROUPID FORMATTING/TRANSFORMATION EXAMPLES

1. The /Formatting/Transformations/WSDL/EBXSnapshots contain three XSLT transformations. These are a single gouping which which is represented in the picture on the right as “WSDL.EBXSnapshots”. The groupId is generated from the folder names.



groupId	process	layerType
WSDL.EBXSnapshots	1	DB
WSDL.EBXSnapshots	1	PM
WSDL.EBXSnapshots	1	FV
WSDL.EBXSnapshots	1	LV
WSDL.EBXSnapshots	1	BV
WSDL.EBXSnapshots	1	CV
WSDL.EBXSnapshots	1	CP
WSDL.EBXSnapshots	1	PV
WSDL.EBXSnapshots	1	CR

2. The /Formatting/Transformations/XML/ds_XML contain one XSLT transformation. This is a single gouping which which is represented in the picture on the right as “XML.ds_XML”. The group id is generated from the folder names.



groupId	process	layerType
XML.ds_XML	1	DB
XML.ds_XML	1	PM
XML.ds_XML	1	FV
XML.ds_XML	1	LV
XML.ds_XML	1	BV
XML.ds_XML	1	CV
XML.ds_XML	1	CP
XML.ds_XML	1	PV
XML.ds_XML	1	CR

3. The following is an ***example of what not do*** in /Formatting/Transformations. Notice how the XML folder contains myTransform1 and then another sub-folder called xform which contains myTransform2. The result of this would be duplicate views being generated because the first groupId "XML" would also result in discovering any sub-folders in that path.

   /Formatting/Transformations/XML/myTransform1

   /Formatting/Transformations/XML/xform/myTransform2

3.3. ***Location***: /<project-path>/_scripts/Configure/ConfigureStartingFolders

3.4. ***Layer Type Designators*** – Layer types are used as a shortcut which contains the knowledge of the source folder and the target folder. To demonstrate how this works in more detail, the "ds_orders1" groupId will be used. For the sake of brevity, the leading project folder path has been removed. The target is shown on top and the source on the bottom so that it is easy to see now the target from one layer becomes the source for the layer above.

DB – Database Views

Target: /services/databases/ds_orders1

Source: /Application/Published/ds_orders1

Application Layer

CP – Client Published

Target: /Application/Published/ds_orders1

Source: /Application/Views/ds_orders1

CR – CRUD (create, read, update, delete) services

Target: /Application/Services/CRUD/ds_inventory

Source: /Physical/Physical/ds_inventory

CV – Client Views

Target: /Application/Views/ds_orders1

Source: /Business/Business/ds_orders1

Business Layer

BV – Business Views

Target: /Business/Business/ds_orders1

Source: /Business/Logical/ds_orders1

LV – Logical Views

Target: /Business/Logical/ds_orders1

Source: /Physical/Formatting/ds_orders1

Physical Layer

FV – Formatting Views

Target: /Physical/Formatting/ds_orders1

Source: /Physical/Metadata/MysqlSource/ORDERS1/ds_orders1

PV – Physical Views

Target: /Physical/Physical/ds_orders1

Source: /Physical/Metadata/MysqlSource/ORDERS1/ds_orders1

PM – Physical Metadata

Target: /Physical/Metadata/MysqlSource/ORDERS1/ds_orders1

Source: /Physical/Metadata

Explicit path or ConfigureStartingFolders

4. Explicit path or ConfigureStartingFolders

4.1. The generate…Views procedures now provide the ability to choose between naming the explicit folder paths for the source and target or using the traditional ConfigureStartingFolders with the layerType, groupId and derivedFilterPath.

4.1.1. [OPTION1] Explicit Folder parameters:

1. publishToFolder

   This is the full path to the folder in which to generate the views

   This is only required if option 1: targetResource is provided.

   If targetResource is not blank, then it is used and groupIds and derivedFilterPath are ignored

2. sourceResource

   The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder

   If this is set it supercedes layerType, inGroupIDs and derivedFilterPath

4.1.2. [OPTION 2] ConfigureStartingFolder parameters:

If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional. The actual inputs are defined as local variables with DEFAULT values. Adjust those variables accordingly.

1. layerType

   layerType is set within the internal parameters for the specific generate…Views procedures.

2. groupIds

   This is a comma-separated list of group ids to process.

   This is a filter that allows the user to only generate for a specific group or list of groups found in the /_scripts/Configure/ConfigureStartingFolders.

   Pass in null to select all groupIds.

3. derivedFilterPath

   The path is derived by concatenating the partial filter path with the source path of the designated layer type. The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. The higher up the folder chain you specify in ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath.

   Exmple:

   layerType=CR

   sourceFolderPath= /shared/ASAssets/BestPractices/DataAbstractionSample/Physical/Physical/ds_orders

   derivedFilterPath=customers,orders

   Even though there are several other views in the /Orders folder under the Application/Views, only the ones specified in the filter path will be generated to the Application/Published. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type.

   Pairing: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter.

   For example:

- groupIds=ds_orders1,ds_orders2
- derivedFilterPath="customers,orders",orders

The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2.

sourceResource: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a CONTAINER/FOLDER resource. If sourceResource points to a TABLE/VIEW resource then derivedFilterPath is ignored.

For example:

- sourceResource=/shared/lab00/Physical/Metadata/ds_orders1
- derivedFilterPath="customers,orders"

The result for the above is that customers and orders are the only views generated.

Note: exactMatch is defaulted to 0 so a partial path match is the only thing required

Resource Annotations and Logical Definitions in the spreadsheet

5. How to provide annotations with tabs and line breaks

5.1. If you are editing the Common_Model_v3_file[1-3].xlsx by hand, you may want to provide annotations in the "Logical Definition" column. You can insert tab characters and line feed characters for new lines. The traditional carriage return is not used in Composite.

5.1.1. <TAB> - this represents a tab character

5.1.2. <LF> - this represents a new line or line feed character in Composite

5.2. When the Best Practices scripts generate the Common Model spreadsheet it will also do the reverse for annotations that it finds in the resources. It will insert <TAB> and <LF> characters where necessary.

Upgrading a Project

6. How to upgrade a project

6.1. This is a new capability in 7.1. The purpose of the upgrade procedure is to assist the user in automatically performing the upgrade of a project from one version of the Best Practices to the latest.

6.2. What can change:

6.2.1. **Generate** – The /Generate folder procedures can change. In this case, the new procedures are copied from the

"DataAbstraction_GENERIC_Template"/_scripts/Generate folder to your project folder and rebound to your project defaultValues constants.

6.2.2. **Custom** – Any procedures that are found in the /Generate folder that do on occur in the "DataAbstraction_GENERIC_Template"/_scripts/Generate are moved to the /Custom folder.

6.2.3. **ConfigureStartingFolders** – From time-to-time, it may be necessary to add variables, set statements, or insert statements to an existing /Configure/ConfigureStartingFolders procedure. These changes are made in-place by replacing text, replacing after text or replacing before text.

6.2.4. **defaultValues** – From time-to-time, it may be necessary to add variables or set statements to the existing /Constants/defaultValues procedures. These changes are made in-place by replacing text, replacing after text or replacing before text.

6.2.5. **Adding new procedures** – From time-to-time, it may be necessary to add new procedures to a folder.

6.2.6. **Deleting procedures** – From time-to-time, it may be necessary to delete a procedure that is no longer being used.

6.2.7. **Moving procedures** – From time-to-time, it may be necessary to move a procedure to a different folder.

6.3. Version Differences and Mappings

6.3.1. Review the section "**Best Practices Version Differences**" for more details on what the differences are between a version and the 7.x/8.x baseline.

6.3.2. Review the section "**Best Practices Version Mappings**" for more details on what the mappings are between a version and the 7.x/8.x baseline.

6.4. Upgrade steps

6.4.1. Review the section "**How to Upgrade the Best Practices Scripts**" for more details on how to perform an upgrade to the latest version.

Max Request Depth

7. The scripts contain a number of recursive procedures. It is recommended to set the Max Request Depth to 100. This value specifies an upper limit on the depth of the request stack in a transaction.

7.1. From Studio, Click: Administration → Configuration → Composite Server → Configuration → Transactions → Max Reques Depth = 100

Debug Settings

8. The scripts contain a number of debug options. Some settings are scoped per project while others are scoped for the entire Best Practices installation.

    8.1. Project Scoped

        8.1.1. Debug Time for Generate Views

            1. Debug Location: ../<project>/_scripts/Constants/defaultValues

                8.1.1.1.1. debugTime – debug time output only

        8.1.2. Debug Generate Views and Generate Display

            1. Debug Location: ../<project>/_scripts/Configure/ConfigureParams

                8.1.2.1.1. debug1 – debug 1st level scripts

                8.1.2.1.2. debug2 – debug 2nd level scripts

                8.1.2.1.3. debug3 – debug 3rd level scripts

        8.1.3. Debug Generate CRUD

            1. Debug Location: ../<project>/_scripts/Constants/defaultValues

                8.1.3.1.1. debug – debug CRUD scripts

                8.1.3.1.2. debugTime – debug CRUD execution time

                8.1.3.1.3. debugException –x debug CRUD exceptions

    8.2. Installation Scoped

        8.2.1. Debug Parse SQL Script

            1. Debug Location: /shared/ASAssets/BestPractices_v81/_ProjectMaintenance/defaultValues

                8.2.1.1.1. debugSqlParser1 – debug 1st (top) level sql parser methods

                8.2.1.1.2. debugSqlParser2 – debug 2nd level sql parser methods

                8.2.1.1.3. debugSqlParser3 – debug 3rd (lowest) level sql parser methods

                8.2.1.1.4. debugTime – debug time output only

How to Change a Windows Service Name and Description

9. The section discusses how to change the service name for Windows. This is useful after upgrading an instance of Composite and you would like to have the service name reflect the version that you are running.

    9.1. Change the service description.

        9.1.1. Open a windows command line that has administrative rights on the computer.

        9.1.2. This examples shows how to change a Windows service description for a windows service from “Composite Server 6.2.0 (3)” to “Composite Server 6.2.5”

        9.1.3. **sc description** "Composite Server 6.2.0 (3)" "Composite Server 6.2.5"

    9.2. Change the service name.

        9.2.1. Open a windows command line that has administrative rights on the computer.

        9.2.2. This examples shows how to change a Windows service name for a windows service from “Composite Server 6.2.0 (3)” to “Composite Server 6.2.5”

        9.2.3. **sc config** "Composite Server 6.2.0 (3)" **DisplayName=** "Composite Server 6.2.5"

            1. Note: There is a space between the equal sign and the value.

                i. Note: Service names with spaces are enclosed in double quotes.

3 Creating a Project

How to create a project from the template

This section discusses the steps for creating a new project folder from the Best Practices template. The Best Practices template is called **DataAbstraction_GENERIC_Template.** The following are a high-level checklist for the creation of a project:

1. Create the project – automated via generateProject()
2. Manually add data sources and XML transformations (as needed)
3. Configure Starting Folders – automated via generateConfigureStartingFolders()
4. Manually edit physical to logical mappings in the Common Model spreadsheet
5. Generate Formatting views – automated via generateFormattingViews()

Create Project [AUTOMATED]

Follow the steps below to create a new project.

1. **Create a new project** – Create and configure the new project.
   a. Expand the folder **/shared/ASAssets/BestPractices_vXX/_ProjectMaintenance**
      1.a.1. Open generateProject(projectPath, generateTestFolder, overwrite)
   b. Click Execute  and enter the following parameters
      1.b.1. **projectPath**: type in your full project path

      e.g. /shared/myProject

      1.b.2. **generateTestFolder**: 1 or [0 or null]

      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.

      0 or null=no (default), 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.

      1.b.3. **Overwrite**: 1 or [o or null]

      1=yes, overwrite the target folder

      0 or null=no (default), do not overwrite the target folder.

      1.b.4. Click refresh  when the procedure finishes to refresh Studio.

1.b.5. Note: this procedure automatically performs the following steps:

- Copies the template folder “DataAbstraction_GENERIC_Template” to the path you specify.
- Modifies the “basePath” variable in /shared/myProject/_scripts/Constants/defaultValues.
- Rebinds several procedures to point to your project path resources instead of the default template folder “DataAbstraction_GENERIC_Template”.
- Update /Documentation trigger parameter paths
- Verify paths have been updated
- Generate the Test folder if the user requested it

Add Data Sources [MANUAL]

2. **Add Data Sources** – Add data sources to your project.

   a. The best practice is to add data sources into sub-folders in the folder: /shared/<projectPath>/Physical/Metadata/<ds_folder>. Be sure and create a sub-folder for each data sources. For XML sources, you will need to create an XML to relational transformation. Create the transformations in the folder /shared/<projectPath>/Physical/Formatting/Transformations/<ds_folder>. Be sure to create a sub-folder as the folder name will be used as the groupId in the ConfigureStartingFolders.

Configure Starting Folders [AUTOMATED or MANUAL]

3. **Create “ConfigureStartingFolders” Script** – Create the ConfigureStartingFolders script either using the “generateConfigureStartingFolders()” or by editing the file.

   a. Location: /shared/<projectPath>/_scripts/Configure/ConfigureStartingFolders

   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.

   b. **AUTOMATED**

   3.b.1. Generate the ConfigureStartingFolders procedures based on existing data sources and transformations.

   3.b.2. Expand **/shared/ASAssets/BestPractices_vXX/_ProjectMaintenance**

   3.b.3. Open generateConfigureStartingFolders(projectPath)

   3.b.4. Click Execute  and enter the following parameters

3.b.4.1. projectPath: type in your full project path

3.b.4.2. e.g. /shared/myProject

3.b.5. Click refresh  when the procedure finishes to refresh Studio.

c. **MANUAL**

3.c.1. There are example sections marked by "**-- SECTION:**". Find a section and modify it to suit your physical metadata sources.

3.c.1.1. 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.

3.c.1.2. Modify the folders according to the sources that you have and your folder structure.

3.c.1.3. Each section as shown below will be used for a data source. Example of what you will see:

SECTION: EXAMPLE ORACLE CISORADEMO DATA SOURCE GENERATION FOLDERS

This example shows how to specify the Physical Metadata CIS datasource path for Oracle. Specify the full path all the way down to the Oracle Schema user folder. This is the folder just above the tables. The 6th parameter is left null in the insert statement for 'PV'.

```
Set groupId = 'CISORADEMO';
SET PM_FOLDER=physicalMetadataPath||'/OracleSource/cisOraDemoDS/CISORADEMO';
SET FV_FOLDER=physicalFormattingPath||'/CISORADEMO';
SET LV_FOLDER=businessLogicalPath||'/Orders_ora';
SET BV_FOLDER=businessBusinessPath||'/Orders';
-- 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 Application Client Views from the Logical Views source path specified
INSERT INTO StartingFolderPipe VALUES (groupId,1,'CV','A',LV_FOLDER,CV_FOLDER);
```

Edit Common Model Spreadsheet [MANUAL or AUTOMATED]

4. **Edit Data Abstraction Best Practices Spreadsheet** – This is used to mapping the physical to the logical table/column names.

a. **MANUAL**

4.a.1. The spreadsheet is called **"Common_Model_v3_file[1-3].xlsx"** and is put in D:/DV/BestPractices.

4.a.2. See the section "Data Dictionary Spreadsheet" for more information.

b. **AUTOMATED**

4.b.1. After you have been working for a while and making edits to the Formatting layer, you may want to synchronize the "**Common_Model_v3_file[1-3].xlsx"** spreadsheet. Use the generateDatasourceListCSV() method to generate a .CSV formatted file. Copy columns A-L into the aforementioned Excel (.xlsx) formatted spreadsheet.

Generate Views [AUTOMATED]

5. **Generate Views** – Generate the views.

a. Views generation disclaimer:

5.a.1. The view generation scripts listed below contains a parameter "**generateViewsWrapper**" that allows the user to toggle between printing the output to the cursor or to the console window. The parameter replaces the specialty procedures that used to exist [generateFormattingViewsWrapper and generatePhysicalViewsWrapper].

- 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply.
- 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.

5.a.1.1. List of procedures containing the parameter "generateViewsWrapper":

- generateViews()
- generateFormattingViews()
- generateLogicalViews()
- generateBusinessViews()
- generateClientViews()
- generateClientPublished()
- generatePublishedResource()
- generatePhysicalViews()
- generateCRUDOperations()

5.a.2. The following scripts are not bound by the "Fetch Row Size" or "Cursor Fetch Limit"

- generateDatasourceListCSV()

- generateTypeDefinitions()

5.a.3. The following script is affected by the Fetch Limit Size and does not contain the parameter “generateViewsWrapper””

- generateDatasourceList()
- generateDatasourceListInsertDB()

b. You are now ready to generate views. Generate in this order

5.b.1. generateFormattingViews(generateViewsWrapper, overwrite, copyAnnotation, copyPrivilegeMode, exactMatch, derivedFilterPath, excludeDsPathsList, sourceResource, generateToFolder, groupIds) – these procedures are used to generate the Physical/Formatting sub-layer views.

5.b.1.1. Review the section “Physical to Logical Mappings” if you want to generate the physical to logical mappings for each physical metadata column and table. The “Formatting” sub-layer is the place to establish how physical names are mapped to “Logical/Canonical” names for both tables and columns. It is recommend performing the mapping prior to Logical views being constructed so as to alleviate any re-work later on if you decide to change the table/column names.

5.b.2. **Optional** views that can be generated:

5.b.2.1. generateLogicalViews(generateViewsWrapper, overwrite, copyAnnotation, copyPrivilegeMode, exactMatch, derivedFilterPath, excludeDsPathsList, sourceResource, generateToFolder, groupIds) – this procedure may be used to generate from the Business/Logical sub-layer into the Business Logical sub-layer. It can only generate simple views. It cannot generate views with multi-table joins.

5.b.2.2. generateBusinessViews(generateViewsWrapper, overwrite, copyAnnotation, copyPrivilegeMode, exactMatch, derivedFilterPath, excludeDsPathsList, sourceResource, generateToFolder, groupIds) – this procedure may be used to generate from the Busines/Business sub-layer into the Business Logical sub-layer. It can only generate simple views. It cannot generate views with multi-table joins.

5.b.2.3. generateClientViews(generateViewsWrapper, overwrite, copyAnnotation, copyPrivilegeMode, exactMatch, derivedFilterPath, excludeDsPathsList, sourceResource, generateToFolder, groupIds) – this procedure may be used to generate views into the Application/Views (client) views sub-layer. It can only generate simple views and not multi-table joins.

5.b.2.4. generatePublishedViews(generateViewsWrapper, overwrite, copyAnnotation, copyPrivilegeMode, exactMatch, derivedFilterPath, excludeDsPathsList, sourceResource, generateToFolder, groupIds) – this procedure may be used to generate views into the Application/Published (client) views sub-layer. It can only generate simple views and not multi-table joins.

5.b.2.5. generatePublishedResource(generateViewsWrapper, overwrite, copyAnnotation, copyPrivilegeMode, exactMatch, derivedFilterPath, excludeDsPathsList, sourceResource, generateToFolder, groupIds) – this procedure may be used to generate LINKS from the Application/Published to the Composite Database folder.

5.b.2.6. generatePhysicalViews(generateViewsWrapper, overwrite, copyAnnotation, copyPrivilegeMode, exactMatch, derivedFilterPath, excludeDsPathsList, sourceResource, generateToFolder, groupIds) – This procedure has been deprecated as a result of the Formatting views subsuming the layer that maps the physical metadata to the formatting sub-layer. However, this procedure is left here for backwards compatibility as well as providing the ability for CRUD procedures to be generated off of physical views that provides a one-to-one mapping with the physical metadata.

5.b.2.7. generateCRUDOperation(generateViewsWrapper, overwrite, copyPrivilegeMode, exactMatch, derivedFilterPath, typeDefProcName, sourceResource, generateToFolder, layerType, groupIds) – Only necessary if you are performing Create, Update, or Delete operations on the generated views. Only for a single database source at this time.

5.b.2.8. generateTypeDefinitions(generateViewsWrapper, overwrite, copyPrivilegeMode, exactMatch, derivedFilterPath, typeDefProcName, sourceResource, generateToFolder, layerType, groupIds) – This is performed automatically when generateCRUDOperation() is executed. It is provided as an independent procedure as well. Unless specified, this script will generate and overwrite any existing procedures named “TypeDefinitionsGen”. The user may wish to generate “public” type definitions for other procedures to use whereby the TypeDefinitionsGen procedure becomes the central location for the definition which all procedures can use.

5.b.2.9. generateViews(various parameters…) – This is the generic API form of the more specific generate-“Layer”-Views(). If the generate-“Layer”-Views() is too confining for your purposes, then you can configure and use generateViews() in a more generic way. It could also be invoked via other procedures to automate the process. It can only generate simple views with no multi-table joins.

6. **Build Views** – Development begins with developers creating Logical resources, Business resources, Application resources and then publishing them as data services to databases and web services.

4 Data Dictionary Spreadsheet

Physical to Logical Mappings

There are three Excel spreadsheets that provide the user the ability to define the mappings for physical to logical names and one Excel spreadsheet that serves the DataAbstractionSample and Labs. These spreadsheets are used to track the data dictionary for one or more projects. The logical mappings are known as the Common Model. Another term often used is the canonical model.

- The first file, Common_Model_v3_file1.xlsx is blank and is available for mappings to be placed into it.
- The second file, Common_Model_v3_file2.xlsx is blank and is available for mappings to be placed into it.
- The third file, Common_Model_v3_file3.xlsx is blank and is available for mappings to be placed into it.
- The fourth file, Common_Model_v3_file4_sample_lab.xlsx contains the Composite sample mappings for the DataAbstractionSample and labs.

Edit Mappings – Columns [A-L]

Edit the Excel (.xlsx) files to enter mappings. The following columns are edited by the user. The following screen shot shows Columns A-L and Column M for System Messages:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Data Source	Project Folder Name	Great Grandparent Name	Grandparent Name	Parent Name	Container Name	Physical Name	Physical Type	Logical Name	Logical Type	Logical Transformation	Logical Definition	System Messages
2	Common_Model_v2	DataAbstractionSample				Common_Model_v2	"Common_Model_v2_file1.csv	TABLE	Common_Model_1	TABLE			
3	Common_Model_v2	DataAbstractionSample			Common_Model_v2	"Common_Model_v2_file1.csv"	"Data Source"	VARCHAR	Data_Source	VARCHAR(50)			
4	Common_Model_v2	DataAbstractionSample			Common_Model_v2	"Common_Model_v2_file1.csv"	"Parent Name"	VARCHAR	Parent_Name	VARCHAR(50)			
5	Common_Model_v2	DataAbstractionSample			Common_Model_v2	"Common_Model_v2_file1.csv"	"Container Name"	VARCHAR	Container_Name	VARCHAR(50)			
6	Common_Model_v2	DataAbstractionSample			Common_Model_v2	"Common_Model_v2_file1.csv"	"Physical Name"	VARCHAR	Physical_Name	VARCHAR(50)			

1. **[col A] Data Source** – (optional) This column indicates what data source the physical attribute is coming from. It is here for documentation purposes.

   1.1. Column A →[maps to] Column N (datasourceName)

2. **[col B] Project Folder Name** – (mandatory) This column provides the name of the **project**. Introducing the name of the project helps to uniquely identify a set of columns that may exist for a given data source across projects.

   2.1. Column B →[maps to] Column O (projectFolderName)

3. **[col C] Great Grand Parent Name** – (optional/mandatory) This column provides the great grand parent name of the project. It can also be considered the **data source name** for the physical resource name. It is required that this name be unique within the project folder even if the data source is separated from like data sources by a folder. If it is not unique, there is no guarantee the scripts will be able to query and locate the correct physical to logical attribute mapping.

3.1. Column C →[maps to] Column P (greatGrandParentName)

4. **[col D] Grand Parent Name** – (optional/mandatory) This column provides the grand parent (grandparent) of the actual resource. It can also be considered the **catalog** for the physical resource name.

   4.1. Column D →[maps to] Column Q (grandParentName)

5. **[col E] Parent Name** – (optional/mandatory) This column provides the parent (parentName) of the actual resource. It can also be considered the **schema** for the physical resource name.

   5.1. Column E →[maps to] Column R (parentName)

6. **[col F] Container Name** – (mandatory) Provides 2 purposes. It can also be considered the **table** for the physical resource name.

   6.1. (1) Provides a way to associate a physical container name such a physical view folder name or schema name to the "Table Name". (2) Provides a way to associate a "Table Name" to the "Table Column Name".

   6.2. Note: If generating from the Formatting Views, then the Parent Container is the Formatting View folder name.

   6.3. Column F →[maps to] Column S (containerName)

7. **[col G] Physical Name** – (mandatory/optional) It can also be considered the physical **column/attribute** for the physical resource name. Identifies the physical table name or column name. If this column is identifying the physical table name, then it is on the same row as the logical table name. If is it identifying physical column names, then the list of column names starts directly below the corresponding physical table name. This column is optional when creating a new derived Logical Name that has no Physical Name association.

   7.1. Column G →[maps to] Column T (physicalName)

8. **[col H] Physical Type** – (recommended) Provide the physical/native type of the resource. It is not used to generate any columns for Composite usage.

   8.1. **Column H** →[maps to] Column U (physicalType)

      8.1.1. Will be TABLE or PROCEDURE when the row represents the resource, otherwise it will be the actuall physical attribute type.

9. **[col I] Logical Name** – (mandatory) Provide the mapping from the physical column name to the logical attribute name/column name. This field is used to calculate Column P.

   9.1. Provides the mapping from the physical table name to the logical table name. Only the row with the physical table name contains a value. It is used to calculate column U, the Composite "logicalName". If no value is provided, the logicalname is left blank; however, the view will be generated with the physical name.

   9.2. **Column I** →[maps to] Column V (resourceName)

10. **[col J] Logical Type** – (recommended) Provide the logical type which is used to generate the cast statements that surround the logical column name and any transformations that exist.

    10.1. **Column J** →[maps to] Column W (logicalType)

11. **[col K] Logical Transformation** – (optional) Contain any transformations that are needed. The transformations are Composite SQL syntax.

11.1. This field automatically prepends a cast(? AS <logicalType>) if col G ("Logical Type") is present.

11.2. **Column K** →[maps to] Column X (compositeTransformation)

12. **[col L] Logical Definition** – (optional) This column provides a definition of the logical entity/attribute. It is for documentation purposes only and is not used to calculate any composite columns.

12.1. **Column L** →[maps to] Column Y (annotation)

13. **[col M] System Messages** – This field provides system messages and warnings

13.1. **ERROR**: Logical Type required – this message occurs when the "Logical Name" and the "Logical Type" are both blank. The "Physical Name" must be blank when creating a new field or derived field that is not in the Physical Metadata. Therefore, there is no known type for a new or derived field. It is required to have a "Logical Type" and "Logical Name" for a derived field. Resolution: add the "Logical Type"

13.2. **WARN**: Logical Name exceeds 28 char – this message occurs when the "Logical Name" exceeds a length of 28 characters. It is a warning because it is only a problem if you try and cache a view to Oracle. Composite will add double quotes around a column name so the real length is 30 characters. Exceeding this will cause Oracle to cut off the names when it creates them. This will cause downstream problems when other views go to access that view.

Composite Logical Transformations – Column [K]

There are different strategies for creating composite transformations as shown below:

1. *Data type casting* – Data type casting is a form of transformation whereby the type of the physical column is cast to a different type as in "cast (FR_CHRG as numeric(12,2)) FreightCharge".

2. *Simple derived columns* – Derived columns are typically columns that can be calculated from existing columns. In the example provided above, the CostPerWeight is calculated from the Freight Charge and the container Weight. Another example would be the concatenation of two or more fields to create a derived column. E.g. CAST(FNAME || LNAME AS VARCHAR)

3. *Value formatting* – Value formatting provides conditional logic to return a different value in place of the original value. An example would be to asses an ID field and return a description. E.g.

```
CASE DESC
  WHEN 'desc1'  THEN 'desc'
  WHEN 'desc2' THEN 'desc'
  ELSE DESC
END AS genericDesc
```

4. *New Fields* – A new column is one in which it does not exist in the dta source. It may be statically defined, derived from other columns in the same table or returned as a result of a custom or system function call. A new field can be introduced into the spreadsheet using the following setup rules:

4.1. Insert a new row at the end of the physical table that you want to associate the new column with. Make sure you copy the formulas.
4.2. Provide [col I] Logical Attribute Name.
4.3. Provide [col J] Composite Transformation
4.4. Do not provide a Physical Column Name [col G] as it is not applicable.
4.5. The following are examples
    4.5.1. Data Source Type – CAST('DS1' as VARCHAR)
    4.5.2. Current Date – CAST(CURRENT_DATE as DATE)
4.6. It is very important that a "Logical Name" name [col E] be present and the "Physical Name" [col G] be blank for this use case.

5. *Null mapping* – It may be necessary to establish the alias column in this layer, yet it has no corresponding physical data element to map to. In this case, it is permissible to map the alias to a NULL value. However, it is also necessary to cast the null to a specific type. E.g. CAST(NULL AS VARCHAR).

6. *Light Data Quality* – Cleaning up known bad data. This use case will most likely use case statements to test value conditions and provide alternative data values.

Composite Generation Script Lookup columns – Columns [N-Y]

The spreadsheet will produce the columns that are used as a "lookup table" by Composite generation scripts. A couple of fields are provided as reference but are not used. There is the set of fields used to query the spreadsheet in order to do a lookup of the physical to logical mapping. There are fields that are used during the generation. The lookup columns from the spreadsheet are as follows:

Not used: [col N] datasourceName
Query: [col O] projectFolderName
Query: [col P] greatGrandParentName
Query: [col Q] grandParentName
Query: [col R] parentName
Qyery: [col S] containerName
Query: [col T] physicalName
Not used: [col U] physicalType
Qyery: [col V] logicalName
Used: [col W] logicalType
Used: [col X] compositeTransformation
Used: [col Y] annotation

	N	O	P	Q	R	S	T	U	V	W	X	Y
1	datasourceName	projectFolderName	greatGrandparentName	grandparentName	parentName	containerName	physicalName	physicalType	logicalName	logicalType	compositeTransformation	annotation
2	Common_Model_v2	DataAbstractionSample				Common_Model_v2	"Common_Model_v2_fil	TABLE	Common_Model_1	TABLE	CAST(? AS TABLE) AS	
3	Common_Model_v2	DataAbstractionSample			Common_Model_v	"Common_Model_v2_	"Data Source"	VARCHAR	Data_Source	VARCHAR(50)	CAST(? AS VARCHAR(50)) AS	
4	Common_Model_v2	DataAbstractionSample			Common_Model_v	"Common_Model_v2_	"Parent Name"	VARCHAR	Parent_Name	VARCHAR(50)	CAST(? AS VARCHAR(50)) AS	
5	Common_Model_v2	DataAbstractionSample			Common_Model_v	"Common_Model_v2_	"Container Name"	VARCHAR	Container_Name	VARCHAR(50)	CAST(? AS VARCHAR(50)) AS	
6	Common_Model_v2	DataAbstractionSample			Common_Model_v	"Common_Model_v2_	"Physical Name"	VARCHAR	Physical_Name	VARCHAR(50)	CAST(? AS VARCHAR(50)) AS	

1. **[col N] datasourceName** – The Composite data source name associated by lineage to the resource.

2. **[col O] projectFolderNme** – This is the project folder name and is represented by last name name found in the defaultValues.basePath under the "/_scripts/Constants" directory for each project.

3. **[col P] greatGrandParentName** – This column provides the parent (grandparent) of the actual resource. For the formatting views and physical views, the parent is actually the parent folder to the view. When the physical resource is a member of a datasource then this may be a "**data source name**" name otherwise it is the great grand parent folder name for the Composite view or procedure.

4. **[col Q] grandParentName** – This column provides the parent (grandparent) of the actual resource. For the formatting views and physical views, the parent is actually the parent folder to the view. When the physical resource is a member of a datasource then this may be a "**catalog**" name otherwise it is the grand parent folder name for the Composite view or procedure.

5. **[col R] parentName** – This column provides the parent (grandparent) of the actual resource. For the formatting views and physical views, the parent is actually the parent folder to the view. When the physical resource is a member of a datasource then this may be a "**schema**" name otherwise it is the parent folder name for the Composite view or procedure.

6. **[col S] containerName** – Container refers to the physical object that contains another object. For example a folder can contain another folder. A Physical **Metadata table** can contain column names. Alternatively the container may be a Physical View name. It all depends on the perspective from which the spreadsheet is generating from. The code will use the container name as a lookup so it is important to get these names exact.

7. **[col T] physicalName** – This is the name of the resource object (**container or column name**). The generation scripts will use the physical resourceName to do the lookup against the spreadsheet.

8. **[col U] physicalType** – The data type of the physical resource. When the physical resource is a member of a data source then this type is the native data type otherwise it will be a Composite type for Views and Procedures.

9. **[col V] logicalName** – The logicalName is the name that you want to transform the physical object into. In Composite Views this is also known as the "**alias**" name.

10. **[col W] logicalType** – The logical column type which is used for casting purposes. If the logical Type is not specified then no CAST statement is generated. A logicalType must be provided for new derived columns.

11. **[col X] compositeTransformation** – The compositeTransformation is used to perform several different types of column level transformations.

12. **[col Y] annotation** – provides both resource-level (table) and column annotations.

Saving the Common Model Spreadsheet

This section describes how to save the common model file spreadsheet.

1. **Excel** – The Excel files are read directly from the Best Practices scripts and cached in MySQL database. Any time the Excel files are modified, simply save them and execute the cache refresh.

2. **CSV Alternative – Saving the Mapping files**.

   2.1. If CSV files are required to be used, the defaultValues.commonModelType must be set to CSV. Location: /shared/ASAssets/BestPractices_v81/_ProjectMaintenance/defaultValues

```
Use Excel as the default source for the Common_Model_V3_file[1-3].xlsx spreadsheet unless CSV is specified
--          'EXCEL' - use the Excel worksheets (default)
--          'CSV'     - use the CSV worksheets
DECLARE PUBLIC commonModelType      CONSTANT VARCHAR DEFAULT 'EXCEL';
```

   2.2. Save the file so that the .xlsx is saved

   2.3. Select File → Save as and

      2.3.1. select “Save as type: CSV (Comma delimited) *.csv” from the list,

      2.3.2. click Save,

      2.3.3. Click Yes to overwrite the existing file,

      2.3.4. Click Yes to keep the this format,

      2.3.5. Close the file and Click No to the question

Check Physical to Logical Mapping Differences

This section describes how to use the **Common_Model_v3_Difference_Phys_to_Log.xlsx** spreadsheet to check differences between the physical and logical mappings.

Common_Model_v3_Difference_Phys_to_Log.xlsx

1. Use this spreadsheet to identify major differences in the physical to logical mappings. This file is used to test the differences between the physical and logical names as well as the physical and logical types.

2. The formulas in the two columns "Diff Name-Physical to Logical" and "Diff Type-Physical to Logical" test for differences.

3. The column "Diff Type-Physical to Logical" in particular only shows differences for types that do not provide a Composite mapping from the physical to the logical types.

5 Best Practices Folder Contents

Folder Contents: /shared/ASAssets/BestPractices_vXX

The contents of the BestPractices folder structure are described the following paragraphs.

Contents: /DataAbstraction_GENERIC_Template

This folder provides a generic template for Composite Data Abstraction Best Practices that can be copied to a project folder. The procedures contained herein are interface procedures only and contain no business logic. Review the _README file in the folder for Input/Output definitions.

1. **/_scripts**

    1.1. **/Configure –** Configure the generation scripts with a list of folders (data source / views) to process.

    1.2. **/Constants** – Contains default values and type definitions.

    1.3. **/Display –** Display data lineage for a resource or a listing of resources in a folder.

    1.4. **/Documentation** – Provides an example of how to generate documentation for a project folder. The Best Practices scripts utilizes the Utilities: /shared/ASAssets/Utilities/documentation/constants and getDocumentationDriver().

        When a project is created from the "DataAbstraction_GENERIC_Template", the user will also have the ability to schedule the documentation generation. There are many options for generating documentation. However, you might consider the breaking the documentation down into the following major areas:

            /services/databases/<your_published_DB>
            Application/Published
            Application/Services
            Application/Views
            Business/Business
            Business/Logical

        This allows you to generate documentation at different layers and get a different perspective.

        After creating a new project, the parameters require editing to provide the actual path to the constants folder.

    1.4.1. **README** – Provides an explanation of the strategy and parameters for each trigger shown below.

1.4.2. **constants** – These are default constants used by the documentation procedures. Constants are project specific and therefore they are copied into each project folder.

1.4.3. **documentationTrigger** – Provides a template to schedule the documentation generation. Duplicate this as needed and configure. This provides a capability to execute "documentationDriverWrapper" on a scheduled basis. The procedure is located in the project _scripts/Documentation folder. Since resources are constantly changing, this offers the ability to stay current with the DV resources. The "documentationTrigger" is project specific and therefore it is copied into each project folder and duplicated as needed. The "documentationTrigger" provides a template (example) to use.

Trigger procedure path:

/shared/<project>/_scripts/Documentation/documentationDriverWrapper

Example: /shared/p1/_scripts/Documentation/documentationDriverWrapper

Trigger parameters:

defaultValuesLayerConstantName – The layer type constant name from defaultValues at which to begin generating documentation.

Example: applicationPublishedPath_

Using getConstantV2() the value would be /shared/P1/Application/Published

subLayerPartialPath – '' or The sub-layer path that gets appended to the result of the default values layer path that gets returned from defaultValues

Example: Using the folder example above, If this triggers was going to target a subfolder below /shared/P1/Application/Published the value would be a sub-layer partial path such as /MyViews. Therefore, this trigger would target /shared/P1/Application/Published/MyViews

documentationFileName – The name of the output documentation file. The parent path is defined in the documentation constants. The parent path is concatenated with this value to provide the full path of the file.

Example: doc_BestPractices_DATABASE.txt

If parentPath = c:/temp/P1 then the file name = c:/temp/P1/doc_BestPractices_DATABASE.txt

in_switches – If left null, the default in the constants file will be used if applicable. Provides guidance on what to print for documentation and save files. This is a space separate list with no spaces before or after the equal sign. The format is print_switch=[option1|{default_option2}|option3]

- print_containers=[{none}|all] - print the resource container (folder)
- print_annotations=[none|{all}|nonblank|blank] - print all annotations whether they are blank or not
- print_resource_projections=[none|{all}] - print the resource projections
- print_resources_used=[none|{all}] - print the immediate child resources used by the parent resource
- print_datasource_accessed=[none|{all}] - print the data source accessed list
- print_datasource_lineage=[none|{all}] - print the data source lineage
- print_time=[{no}|yes] - print the time it takes to retrieve the full documentation for each resource and the final time
- save_file=[{no}|yes] - save the results to a file

- save_file_intermediate=[{no}|yes] - save the file intermediately after each resource is completed
- Example:

  1) switches: when left blank or null then the defaults are taken result: all documentation modules are printed

  2) switches: print_annotations=nonblank print_resource_projections=none print_resources_used=none print_datasource_lineage=none

  result: only non-blank annotations are printed and nothing else

in_excludeKeywordsInPathList – If left null, the default in the constants file will be used if applicable. Exclude actual paths. Double quotes are not required. Comma separated list.

in_excludePathsList – If left null, the default in the constants file will be used if applicable. Exclude paths when finding matches for datasources. This is a comma separated list of paths to exclude from processing. It may include any CIS path such as /shared/ASAssets/Utilities, /shared/ASAssets/BestPractices, /lib, /system etc.

1.4.4. **documentationTrigger_DATABASE** – Provides a template to schedule the documentation generation for a Composite Published Database located at /services/databases/<your_published_DB>

Trigger: documentationTrigger_DATABASE -- parameters:

defaultValuesLayerConstantName	compositeDatabasePath_
subLayerPartialPath	'' or any residual path beyond the default.
documentationFileName	doc_BestPractices_DATABASE.txt
in_switches	'' (use the defaults)
in_excludeKeywordsInPathList	'' (use the defaults)
in_excludePathsList	'' (use the defaults)

1.4.5. **documentationTrigger_ Application_Published** – Provides a template to schedule the documentation generation for the Application/Published.

Trigger: documentationTrigger_Application_Published -- parameters:

defaultValuesLayerConstantName	applicationPublishedPath_
subLayerPartialPath	'' or any residual path beyond the default.
documentationFileName	doc_BestPractices_Application_Published.txt
in_switches	'' (use the defaults)
in_excludeKeywordsInPathList	'' (use the defaults)
in_excludePathsList	'' (use the defaults)

1.4.6. **documentationTrigger_ Application_Services** – Provides a template to schedule the documentation generation for the Application/Services.

Trigger: documentationTrigger_Application_Services -- parameters:

defaultValuesLayerConstantName	applicationServicesPath_

subLayerPartialPath	'' or any residual path beyond the default.
documentationFileName	doc_BestPractices_Application_Services.txt
in_switches	'' (use the defaults)
in_excludeKeywordsInPathList	'' (use the defaults)
in_excludePathsList	'' (use the defaults)

1.4.7. **documentationTrigger_Application_Views** – Provides a template to schedule the documentation generation for the Application/Views.

Trigger: documentationTrigger_Application_Views -- parameters:

defaultValuesLayerConstantName	applicationViewsPath_
subLayerPartialPath	'' or any residual path beyond the default.
documentationFileName	doc_BestPractices_Application_Views.txt
in_switches	'' (use the defaults)
in_excludeKeywordsInPathList	'' (use the defaults)
in_excludePathsList	'' (use the defaults)

1.4.8. **documentationTrigger_ Business_Business** – Provides a template to schedule the documentation generation for the Business/Business.

Trigger: documentationTrigger_Business_Business -- parameters:

defaultValuesLayerConstantName	businessBusinessPath_
subLayerPartialPath	'' or any residual path beyond the default.
documentationFileName	doc_BestPractices_Business_Business.txt
in_switches	'' (use the defaults)
in_excludeKeywordsInPathList	'' (use the defaults)
in_excludePathsList	'' (use the defaults)

1.4.9. **documentationTrigger_ Business_Logical** – Provides a template to schedule the documentation generation for the Business/Logical.

Trigger: documentationTrigger_Business_Logical -- parameters:

defaultValuesLayerConstantName	businessLogicalPath_
subLayerPartialPath	'' or any residual path beyond the default.
documentationFileName	doc_BestPractices_Business_Logical.txt
in_switches	'' (use the defaults)
in_excludeKeywordsInPathList	'' (use the defaults)
in_excludePathsList	'' (use the defaults)

1.5. **/Generate –** Generate views, CRUD operations or type definitions.

1.6. **/Rebind–** Rebind all the views from one folder to a different folder.

2. ***Application –*** container to hold client mapping views, procedures and published views

2.1. **DefinitionSets –** provides a place to define definition sets for SQL, Schema and WSDL.

2.2. ***Published* –** client published views have a one-to-one correspondence with the Client views but also contain explicit cast statements which represent the client contract. Contract-first web service implementation stubs are generated here.

2.3. ***Services* –** client procedures either generated or created here. Additionally, procedures are generated here. Note: if generating CRUD procedures, TypeDefinitionsGen()" will be automatically generated and will overwrite any existing procedure named "TypeDefinitionsGen" in the /CRUD/<groupId_path>/Definitions folder.

2.4. ***Views* –** client facing views are created or generated here

3. ***Business* –** container to hold business logical views

3.1. ***Business* –** These views contain "where clauses" to create selective business-oriented views of data or aggregation of data.

3.2. ***Logical* –** Subject areas that are compliant with the customer logical/canonical data model. These views have no "where clauses" since they are general purpose. These views will contain, simple one-to-one with the formatting, simple joins, federated joins and federated (union) of multiple, similar logical views to produce a single, unified result.

4. ***Physical* –** container to hold physical sources and views and first layer of transformation.

4.1. **DiscoveryModels** – Composite Discovery is used to introspect data sources and provide table statistics and relationships in the data. The models that are generated are saved to the DiscoveryModels folder.

4.2. ***Formatting* –** physical to business logical formatting and transformation. Additionally, these views provide an abstraction of the physical sources providing a layer of insulation for rebinding and caching.

4.2.1. ***Transformations* –** transformations will be necessary when mapping XML hierarchical data into rowset data. Typically XSLT scripts are used for this.

4.3. ***Physical* –** physical views are still supported but deprecated in this release. New implementations should use Formatting Views to map physical to logical.

4.4. ***Metadata* –** access to physical source structures (tables, files, web services etc.).

4.4.1. ***SequenceGenerator* –** container to hold the sequence generator façade

4.4.1.1. ***getUniqueID()* –** a facade procedure which is used to invoke the true generator which is implementation specific.

Contents: /DataAbstractionSample

This folder provides a sample in which to practice using the generation scripts. It is based on the out-of-the-box Composite data sources for Customer and Orders. This sample uses the spreadsheet, Common_Model_v3_file4_sample_lab.xlsx and its corresponding .csv file.

Contents: /DataSource

This folder contains the data source for Excel and CSV. There is a dependency on 3 file system Excel and CSV files located in C:/DV/BestPractices/BestPractices\BestPractices_vXX.

- Common_Model_v3_file1.xlsx / .csv
- Common_Model_v3_file2.xlsx / .csv
- Common_Model_v3_file3.xlsx / .csv

Contents: /Procedures

This folder contains common procedures that are the business logic for generating views. Code is not duplicated across projects.

Folder Contents: /shared/ASAssets/Utilities

The Best Practices have a dependency on the /shared/ASAssets/Utilities.

6 Generation Scripts Method Definitions

Detailed Definitions

Detailed documentation on the inputs and outputs can be found in the header and annotation section of each procedure.

Project Maintenance: Display Best Practices Version Script

1. **getBestPracticesVersion –** This procedure returns the version of the Data Abstraction Best Practices. The procedure is located in "/shared/ASAssets/BestPractices_vXX/_ProjectMaintenance".

Direction	Parameter Name	Parameter Type
OUT	bestPracticesVersion	VARCHAR(15)

Project Maintenance: Generate Project

2. **generateProject –** This procedure is used to copy the DataAbstraction_GENERIC_Template to your named project path and modify the resources within the copied project. The procedure is located in "/shared/ASAssets/BestPractices_vXX/_ProjectMaintenance".

   Note: – this procedure automatically performs the following:
   - Copies the template folder "DataAbstraction_GENERIC_Template" to the path you specify.
   - Modifies the "basePath" variable in /shared/<project-path>/Constants/defaultValues.
   - Rebinds several procedures to point to /shared/<project-path> resources instead of the default template folder "DataAbstraction_GENERIC_Template".
   - Update /Documentation trigger parameter paths
   - Verify paths have been updated
   - Generate the Test folder if the user requested it

   If the project path already exists, this procedure will not copy over it unless overwrite=1. This procedure is used to configure a new project with the correct paths.

Direction	Parameter Name	Parameter Type
IN	**projectPath** – the full path to the project that you want to create	VARCHAR(1024)
IN	**generateTestFolder** – determine whether to generate the Test folder or not • 1=generate Test folder, • 0 or null=do not generate Test folder	BIT

Direction	Parameter Name	Parameter Type
IN	**overwrite** – determine whether to overwrite the target project path • 1=overwrite the project • 0=do not overwrite the project	BIT
OUT	**message** – a resulting message	LONGVARCHAR

Project Maintenance: Generate Configure Starting Folders

3. **generateConfigureStartingFolders –** 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. If the ConfigureStartingFolders procedure exists, this method will create a copy in the same folder. The procedure is located in /shared/ASAssets/BestPractices_vXX/_ProjectMaintenance.

Direction	Parameter Name	Parameter Type
IN	**projectPath** – the full path to the project that is to be configured	VARCHAR(1024)
OUT	**message** – a resulting message	LONGVARCHAR

Project Maintenance: Move Project

4. **moveProject –** This procedure is used to move an existing project to another folder structure and correctly update all of the necessary project paths. The procedure is located in /shared/ASAssets/BestPractices_vXX/_ProjectMaintenance. The procedure modifies the following:

    - /shared/<project>/_scripts/Constants/defaultValues.basePath
    - /shared/<project>/_scripts/Documentation/documentationTrigger.*
    - /shared/<project>/_scripts/Generate/generate*
    - All project resources are automatically rebound to the new path.
    - The text for view (SQL_TABLE) and procedure (SQL_SCRIPT_PROCEDURE) resources are searched for text containing the old path and replaced with the new path and then updated in the repository.

    If the project path already exists, this procedure will not copy over it unless overwrite is set to 1 (true).

Direction	Parameter Name	Parameter Type

Direction	Parameter Name	Parameter Type
IN	**oldProjectPath** – the full path to the old project	VARCHAR(1024)
IN	**newProjectPath** – the full path to the new project	VARCHAR(1024)
IN	**generateTestFolder** – determine whether to generate the Test folder or not • 1=generate Test folder, • 0 or null=do not generate Test folder (default)	BIT
IN	**updateScriptText** – determine whether the VIEW or PROCEDURE text should be modified • 1=modify the VIEW or PROCEDURE scrpt text and replace old paths with new paths in the text. • 0 or null=do not modify the VIEW or PROCEDURE script text	BIT
IN	**overwrite** – determine whether to overwrite the new project path • 1=overwrite the project • 0=do not overwrite the project	BIT
OUT	**message** – a resulting message	LONGVARCHAR

Project Maintenance: Rename Project

5. **renameProject –** This procedure is used to rename an existing project within the same folder structure and correctly update all of the necessary project paths. The procedure is located in /shared/ASAssets/BestPractices_vXX/_ProjectMaintenance. The procedure modifies the following:

    - /shared/<project>/_scripts/Constants/defaultValues.basePath
    - /shared/<project>/_scripts/Documentation/documentationTrigger.*
    - /shared/<project>/_scripts/Generate/generate*
    - All project resources are automatically rebound to the new path.
    - The text for view (SQL_TABLE) and procedure (SQL_SCRIPT_PROCEDURE) resources are searched for text containing the old path and replaced with the new path and then updated in the repository.

    If the project path already exists, this procedure will not copy over it unless overwrite is set to 1 (true).

Direction	Parameter Name	Parameter Type
IN	**oldProjectPath** – the full path to the old project	VARCHAR(1024)
IN	**newProjectPath** – the full path to the new project	VARCHAR(1024)
IN	**generateTestFolder** – determine whether to generate the Test folder or not • 1=generate Test folder, • 0 or null=do not generate Test folder (default)	BIT
IN	**updateScriptText** – determine whether the VIEW or PROCEDURE text should be modified • 1=modify the VIEW or PROCEDURE scrpt text and replace old paths with new paths in the text. • 0 or null=do not modify the VIEW or PROCEDURE script text	BIT
IN	**overwrite** – determine whether to overwrite the new project path • 1=overwrite the project • 0=do not overwrite the project	BIT
OUT	**message** – a resulting message	LONGVARCHAR

Project Maintenance: Upgrade Project

6. **upgradeProject –** This procedure is used to upgrade an existing project from one version of the Best Practices scripts to another. Upgrading starts with Version 1.0 and forward. For versions 1.0 through 6.6 a copy of the project is created with the same name and an _vXX post-fixed to the end. The XX represents the current version of the Best Practices. For example, for 7.3 and a project named /shared/MyProject the new project would be called /shared/MyProject_v73. The one caveat to this is that the /services/databases and /services/webervices resources are not rebound to the new project. This will have to be done manually or by using a script. For /services/databases, you can use the "generatePublishedResource" script to re-publish views in bulk from a source folder to a target schema folder. For web services, you will have to rebind them manually.

   For any 7.x based project up to the current version, only the scripts are upgraded since the 7.x baseline folder structure remains unchanged. If a project is 7.x based and is multiple versions behind the current one, the project will be upgraded incrementally from the lowest version found to the version specified by "upgradeToVersion".

   If the "upgradeToVersion" is left null, assume the upgrade is to the current version. The user may also decide to only do a partial upgrade to a specific version as long as all of the current API's support that version. If the API's do not support that version, then a full upgrade will be

required. This will become evident by any impacted resources (resources that are red). Generally speaking, an upgrade is always performed to the latest version.

Upgrades are not incremental. The upgradeProject() script determines the current version and upgrades to the specified version which is typically the latest version. All necessary upgrades tasks are executed within this context.

One of the tasks that is executed during upgrade is to located the copy of the _scripts folder and get the default Values and apply those original values to the current defaultValues.

Note:

- The procedure "**upgradeProjectVersionVector**" contains a vector "**masterUpgradeVector**" of update actions. This vector is maintained by the developer. Specific upgrade actions are maintained for each release. Upgrade actions may include any number of these actions:
    - copy - copy srcResource to dstResource. srcResource and dstResource required. updateStruct is null.
        - if resource type is CONTAINER then copy all resources
        - if resource type is not CONTAINER then copy specific resource
    - copyLeave - copy srcResource to _Copy_#. srcResource is required. dstResource and updateStruct are null.
        - Creates a copy of a resource and leaves the original in place.
        - The copy detects other copies and increments the number as needed in the format of _Copy_#
    - copyRename - copy srcResource to _Copy_#. srcResource is required. dstResource and updateStruct are null.
        - Creates a copy of a resource and renames the original to it. The original is no longer present.
        - The copy detects other copies and increments the number as needed in the format of _Copy_#
    - copyChildren - copy the children of srcResource to dstResource. srcResource and dstResource required. updateStruct is null.
        - Only copying from CONTAINER to CONTAINER is allowed.
    - update - update dstResource using updateStruct. srcResource is null.
    - updateTrigger - update all trigger resources found starting at dstResource using updateStruct. srcResource is null.
    - dstResource can be a single trigger or folder of triggers.
    - updateCrud - update CRUD resources specified by dstResource using updateStruct. srcResource is null.
    - delete - delete dstResource. srcResource and updateStruct are null.
    - move - move srcResource to dstResource. updateStruct is null.

- moveCustom - move all custom scripts from srcResource folder to the specified dstResource folder. updateStruct is null.
  - a custom resource is determined by finding a resource in dstResource that is not in the DataAbstraction_GENERIC_Template.
- rebind - rebind srcResource to dstResource using startingFolder. updateStruct is null.
- The location of “vector_maserUpgradeVector” is as follows: “/shared/ASAssets/BestPractices_vXX/Procedures/generateConfigure/vector_maserUpgradeVector”

- The procedure “**upgradeProject**” uses another vector “**vector_upgradeVersionVector**” of releases. The "vector_upgradeVersionVector" provides a complete list of all the upgrade paths over time starting with 7.0 to 7.1. The "vector_upgradeVersionVector" is like a linked list. The last "bestPracticesVersionTo" becomes the "bestPracticesVersionFrom" for the next pair of versions. This is how the list is used to upgrade from one version to the next. If there is any break in this sequence, then the upgrade stops. It is the responsibility of the developer to add new upgrade path entries for each new release of the Best Practices scripts. An example vector is shown below:

```
DECLARE upgradeVersionVector VECTOR(ROW(bestPracticesVersionFrom DOUBLE,
bestPracticesVersionTo DOUBLE, backupType INTEGER))
 DEFAULT VECTOR[
    (1.0, bestPracticesVersionDefault, 1)-- upgrade 1.0 - 4.0 to current version (backup entire folder)
    ,(5.0, bestPracticesVersionDefault, 1)-- upgrade 5.0 - 5.2 to current version (backup entire folder)
    ,(6.0, bestPracticesVersionDefault, 1)-- upgrade 6.0 - 6.6 to current version (backup entire folder)
    ,(7.0, bestPracticesVersionDefault, 1)-- upgrade 7.3 to current version    (backup entire folder)
    ,(7.1, bestPracticesVersionDefault, 1)-- upgrade 7.3 to current version    (backup entire folder)
    ,(7.2, bestPracticesVersionDefault, 1)-- upgrade 7.3 to current version    (backup entire folder)
    ,(7.3, bestPracticesVersionDefault, 1)-- upgrade 7.3 to current version    (backup entire folder)
    --etc.];
```

  - For versions 1.0, 5.0 and 6.0 compatible, they can only be upgraded directly to the current version.
  - In the above vector, there is a clear upgrade path from 7.0 to 7.1 followed by 7.1 to 7.2 and finally 7.2 to 7.3. Each upgrade “To” version is linked to the next “From” version.
  - The location of “upgradeProject” is as follows: “/shared/ASAssets/BestPractices_vXX/Procedures/generateConfigure/upgradeProject”

Direction	Parameter Name	Parameter Type
IN	**upgradeToVersion** - the version to upgrade the project to	DOUBLE
IN	**projectPath** – the full path to the project that you want to create	VARCHAR(1024)
IN	**generateTestFolder** – determine whether to generate the Test folder or not	BIT

Direction	Parameter Name	Parameter Type
	• 1=generate Test folder, • 0 or null=do not generate Test folder (default)	
OUT	**message** – a resulting message	LONGVARCHAR

Project Maintenance: Rebind Generation Scripts

These scripts are used for rebinding the generation scripts from one folder to another.

7. **rebindGenerationScripts** – This procedure is used to rebind the generation scripts "<baseFolder>/"_sripts"/Constants/defaultValues". Since several of the scripts point to this folder to provide basic starting folder information, it is vital that the correct binding be in place. For example, if you were to copy one of the generation scripts from folder A to folder B...this would not change the bindings to the "<baseFolder>/"_scripts"/Consants/defaultValues". You could go in by hand and change each and every script but that is potentially error prone. This script will help you to rebind all of the generation scripts that make a reference to the "rebindOldFolder" and allow you to rebind to the new "rebindNewFolder". The starting point for this script is provided by "startingResourceFolder". For example, let's say that you copied scripts out of /shared/ASAssets/BestPractices_vXX/DataAbstraction_GENERIC_Template and pasted them into /shared/ASAssets/BestPractices_vXX/DataAbstractionSample. The scripts still point to the folder /shared/ASAssets/BestPractices_vXX/DataAbstraction_GENERIC_Template/Constants. The sample values below show what the input should look like to execute the rebinding for all of the generation scripts.

Direction	Parameter Name	Parameter Type
IN	**startingResourceFolder** - the folder "<baseFolder>/_scripts" from which to start from and interrogate all of the resource Views in it. Example: /shared/ASAssets/BestPractices_vXX/DataAbstractionSample/Generate	/shared/ASAssets/Utilities/TypeDefinitions.pathType
IN	**rebindOldFolder** - the old folder "<oldBaseFolder> " which the scripts are currently pointing to. Example: /shared/ASAssets/BestPractices_vXX/DataAbstraction_GENERIC_Template	/shared/ASAssets/Utilities/TypeDefinitions.pathType
IN	**rebindNewFolder** - the new folder "<baseFolder>/" which you want the scripts to point to. Example: /shared/ASAssets/BestPractices_vXX/DataAbstractionSample	/shared/ASAssets/Utilities/TypeDefinitions.pathType

Direction	Parameter Name	Parameter Type
OUT	**result** - 'success' or 'failure'	VARCHAR(255)
OUT	**resultMessage** - null if success or a message text upon failure.	VARCHAR(1024)

Project Maintenance: Generate Views Validation

8. **validategenerateViews –** This procedure is used to compare resources between a source folder and a mirror copy target folder. This is useful when validating if the view generation outcome. For example, it would be useful to perform a validation against the original Formatting views (Formating_Copy_1) and the newly generated Formatting views (Formatting) in order to validate that the generation process was successful.

   There are two modes for comparing resources: S=SQL Script compare and C=Column and type compare. One thing to bare in mind is that the generation scripts will generate a view with CAST statements when a logical type appears in the spreadsheet. By virtue of the default behavior of generateDatasourceListCSV, it will automatically generate Logical Types. Therefore if the original view did not have CAST statements, the comparison using "S=SQLScript compare" will result in the view not being equal. However the same comparison with "C=Column compare" would yield an equal match upon comparison as it simply checking the column name and type combination.

   This procedure is useful when performing the following "**round trip**" scenario:

   1. A spreadsheet of physical to logical mappings has been established

   2. Generate the Formatting layer views
      generateFormattingViews(1, 2, 1, 0, null, null, null, null, null, null)

   3. Make a copy of the Formatting layer views
      Target=/shared/<project_path>/Physical/Formatting_Copy_1

   4. Generate the Formatting layer to a CSV file
      generateDatasourceListCSV(C:\temp\test.csv, 1000, 1, 1, R, null, null, null, null, null, null, null, FV, null)

   5. Copy test.csv to Common_Model_v3_file1.xlsx Excel: C:/DV/BestPractices/BestPractices\BestPractices_v80\Common_Model_v3_file1.xlsx

      a. Open Common_Model_v3_file1.xlsx in Excel

      b. Open test.csv in Excel

      c. Select cell A2 and use the keystrokes: Ctrl-Shift-End (this will select Columns A-L and all rows except the header row

d. Do a Ctrl-C (copy)

e. Switch to Common_Model_v3_file1.xlsx

f. Make sure there is no data in columns A-L and all of the rows. If there is, then either delete it or pick a different spreadsheet: Common_Model_v3_file[1-3].xlsx

g. Place your cursor in cell A2

h. Do a Ctrl-V (Paste)

i. Save the spreadsheet

6. Refresh the cache of the spreadsheets: Common_Model_v3_file[1-3].xlsx

a. Switch to Studio, Manager tab

b. Select "Cached Resources"

c. Select "common_model"

d. Click Refresh Cache

7. Remove the views in the Formatting layer except the "/Transformations" folder which is considered a source folder for transormation procedures.

8. Generate the Formattin glayer views – generateFormattingViews(1, 2, 1, 0, null, null, null, null, null, null)

Direction	Parameter Name	Parameter Type
IN	**sourceFolderPath** - /shared/BestPracticesTest/Physical/Formatting	LONGVARCHAR
IN	**targetFolderPath** - /shared/BestPracticesTest/Physical/Formatting_Copy_1	LONGVARCHAR
IN	**resourceNameFilter** • Pass in a resource name to filter on from the source folder. This can be a comma separated list. • The column name may not contain commas. Do not place double quotes around the name.	VARCHAR
IN	**generateViewsWrapper** • 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to	INTEGER

	console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = Not applicable for this use case.	
IN	**statusMode** - status mode: A (default)=output All status, E=output Equal status, N=output Not equal status	CHAR(1)
IN	**compareMode** - compare mode: S=compare SqlText, C=compare Columns	LONGVARCHAR
IN	**debug** - Y=debug is on, N=do not debug	CHAR(1)
OUT	**result** - /shared/ASAssets/BestPractices_v81/Procedures/ validategenerateViews. validategenerateViewsType PUBLIC TYPE validategenerateViewsType ROW(status VARCHAR(20),-- "SKIPPED", "EQUAL", "NOT EQUAL", "TARGET NOT FOUND" resourceType VARCHAR(10), -- TABLE or PROCEDURE sourceResourcePath LONGVARCHAR, taregetResourcePath LONGVARCHAR);	CURSOR

9. Validate the view generation process
validategenerateViews(/shared/<project_path>/Physical/Formatting, /shared/<project_path>/Physical/Formatting_Copy_1, null, 1, N, C, N)

Project Maintenance: Update Impacted Resources

9. **updateImpactedResources –** In some versions of Composite such as CIS 6.1, there were issues with resources being impacted after being imported. The typical error messages might be "Session may not be null" or "Session is closed". This procedure is used to traverse the "starting folder" folder structure and look for any impacted resource and attempt to fix the impacted resources. It will simply read in the procedure and save it back out which typically resolves the error unless there is an actual syntax error. It does not resolve data sources that require repintrospection.

Direction	Parameter Name	Parameter Type
IN	**inStartingFolders** - a comma separated list of startingFolder paths like: /shared/ASAssets/BestPractices_v81. If left blank, it defaults to /shared/ASAssets/BestPractices_vXX	LONGVARCHAR
IN	**inExcludePathsKeywords** - exclude paths containing case insensitive keywords	LONGVARCHAR
OUT	**success** – 1=success, 0=error	BIT

Project Maintenance: Validate Project Resources

10. **validateProjectResources –** This procedure is used to validate the project resources and confirm that all resources are within the boundaries of the project. The invocation excludes /shared/ASAssets, /system and /lib folders by default. The invocation may also provide an exclusion list of folders such as a folder or folders that serve as common folders for the project but live outside the boundaries of the project. The main objective is to find any resources that will left orphaned before the project is deployed. .

Direction	Parameter Name	Parameter Type
IN	**startingPath** – the starting folder from which to recursively list resources in order to interrogate their used resources. Typically the startingPath and projectPath are identical however, the startingPath may be a subfolder within the project path.	LONGVARCHAR
IN	**projectPath** – the path to the best practices project. This may be the same path as the startingPath.	LONGVARCHAR
IN	**excludePathList** – comma separate list of paths to exclude from the validation check. /shared/ASAssets, /system, /lib are added by default to the list passed in.	LONGVARCHAR
IN	**debug –** Y=debug on, N=debug off	CHAR(1)
OUT	**success** – 1=success, 0=error	BIT

Project Maintenance: Prune Resources

11. **pruneResources –** This procedure is used to retrieve or delete rows from the target folder that are not used by the source folder. Example: The Formatting views contain a subset of the views in Metadata. The objective is to prune the tables in the Metadata that are not being used.

Direction	Parameter Name	Parameter Type
IN	**operation** – R=retrieve, D=delete resulting rows from target.	LONGVARCHAR
IN	**sourcePath** – the source path to the folder that will be analyzed and compared with resources in the target folder to determine which target folder resources should be pruned (deleted).	LONGVARCHAR
IN	**sourceExcludePathList** – comma separated list of paths to exclude.	LONGVARCHAR
IN	**targetPath** – the target path to the folder where resources will be pruned.	LONGVARCHAR
IN	**targetExcludePathList** – comma separated list of paths to exclude.	LONGVARCHAR
IN	**debug –** Y=debug on, N=debug off	CHAR(1)
OUT	**success** – 1=success, 0=error	BIT

Display Scripts: Display Lineage Tree

12. **displayLineageTree –** Display the Data Lineage relationships for a given CIS resource such as a procedure or view.

Direction	Parameter Name	Parameter Type
IN	**resourcePath** – CIS source resource path to being assessing the parent data lineage	VARCHAR(1024)
IN	**inIgnoreResourceDoesNotExist** – Ignore any resources that do not exist Values: • 1 = Bypass the processing of that resource. • 0 = (default) Do not ignore any resources. Throw an error that the resource does not exist.	INTEGER
OUT	**resourceTreeList** RepositoryDefinitionsRecursive.lineageTreeType seqNum INTEGER, -- a unique id generated resourceID INTEGER, -- id of the Composite resource parentID INTEGER, -- row relates to a resource id resDepth INTEGER, -- depth from parend treeType VARCHAR, -- Parent, Child resName VARCHAR, -- the resource name resPath TypeDefinitions.pathType, -- resource path resType VARCHAR, -- the resource type subtype VARCHAR, -- the resource sub type enabled BIT, -- resource enabled or not dsID INTEGER -- data source id dsResName VARCHAR, -- data source name dsResPath TypeDefintions.pathType, -- resource path dsResType VARCHAR, -- data source type dsResSubType VARCHAR, -- data source sub type dsEnabled BIT, -- data source enabled or not dsChildCount INTEGER -- number of children)	CURSOR

Example:

id,parentId,treeType,resDepth,resName,resPath,resType,53eparat

0,[NULL],Parent,0,ViewSupplier,/shared/examples/ViewSupplier,TABLE,SQL_TABLE

1,0,Child,1,inventorytransactions,/shared/examples/ds_inventory/inventorytransactions,TABLE,DATABASE_TABLE

2,0,Child,1,purchaseorders,/shared/examples/ds_inventory/purchaseorders,TABLE,DATABASE_TABLE

3,0,Child,1,suppliers,/shared/examples/ds_inventory/suppliers,TABLE,DATABASE_TABLE

Display Scripts: Display Resource Tree

13. **displayResourceTree –** Display all the resources that reside within a particular folder container.

Direction	Parameter Name	Parameter Type
IN	**resourcePath** – CIS source folder path to begin deriving a list of resources	VARCHAR(1000)
OUT	**result** CURSOR (ResourceName VARCHAR, - name of the CIS resource ResourcePath TypeDefinitions.pathType, - resource path ResourceType VARCHAR, - resource type ResourceSubType VARCHAR – resource sub type)	CURSOR

ResourceName,ResourcePath,ResourceType,ResourceSubType

CompositeView,/shared/examples/CompositeView,TABLE,SQL_TABLE

InventoryTransactions,/shared/examples/InventoryTransactions,DEFINITION_SET,XML_SCHEMA_DEFINITION_SET

LookupProduct,/shared/examples/LookupProduct,PROCEDURE,SQL_SCRIPT_PROCEDURE

ViewOrder,/shared/examples/ViewOrder,TABLE,SQL_TABLE

ViewSales,/shared/examples/ViewSales,TABLE,SQL_TABLE

ViewSupplier,/shared/examples/ViewSupplier,TABLE,SQL_TABLE

ds_XML,/shared/examples/ds_XML,DATA_SOURCE,XML_FILE_DATA_SOURCE

ds_inventory,/shared/examples/ds_inventory,DATA_SOURCE,RELATIONAL_DATA_SOURCE

ds_orders,/shared/examples/ds_orders,DATA_SOURCE,RELATIONAL_DATA_SOURCE

getInventoryTransactions,/shared/examples/getInventoryTransactions,PROCEDURE,XQUERY_TRANSFORM_PROCEDURE

productCatalog_Transformation,/shared/examples/productCatalog_Transformation,PROCEDURE,XSLT_TRANSFORM_PROCEDURE

Display Scripts: Search Resource Tree

14. **searchResourceTree –** Search the resource tree as per the passed in ResourcePath for the passed in ResourceName.

Direction	Parameter Name	Parameter Type
IN	**resourcePath** – CIS source folder path to begin deriving a list of resources	VARCHAR(1000)
IN	**resourceName** – CIS resource name or wild card search.	VARCHAR(255)
IN	**ignoreCase** – Y/T=ignore case. Match on any case N/F/NULL=exact match. Do not ignore case.	CHAR(1)
OUT	**result** CURSOR (ResourceName VARCHAR, - name of the CIS resource ResourcePath TypeDefinitions.pathType, - resource path ResourceType VARCHAR, - resource type ResourceSubType VARCHAR – resource sub type)	CURSOR

View Generation: Generate Composite Database Published Resources

15. **generatePublishedResource –** This procedure is used for generating LINKS into the Composite Database. An entire source folder in CIS can be searched for views to publish to a Composite data source or the specific source View or Procedure to publish to a Composite data source. Procedure resources may also be published. The database path including the database name, catalogs and schemas must already be created or this method will throw an exception.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console	BIT

Direction	Parameter Name	Parameter Type
	window. Do not print to the cursor result window.	
IN	**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"=overwrite the resource if it exists.	INTEGER
IN	**copyAnnotation** - allows user to decide whether they want to copy annotations or not form both resource and columns. • 0 or null (default)=false=do not copy the annotation from the target resource • 1=true=do copy the annotation from the target resource	BIT
IN	**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.	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the “sourceResource+derivedFilterPath” is contained within the resourcePath. In this case “/shared/project/Physical/Metadata/ds_orders1/order”	BIT

Direction	Parameter Name	Parameter Type
	is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders + the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure.	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	Option 1 In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. Option 2 The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points	

Direction	Parameter Name	Parameter Type
	to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ds_orders1 • derivedFilterPath="customers,orders" The result for the above is that customers and orders are the only views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	LONGVARCHAR
OPTION 1	**Explicit Folders:** Use explicit sourceResource and publishToFolder parameters with derivedFilterPath being optional.	
IN	**sourceResource –** The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder. If this is set it supercedes layerType, inGroupIDs and derivedFilterPath	
IN	**publishToFolder –**This is the full path to the folder in which to generate the views. This is only required if option 1: targetResource is provided. If sourceResource is not blank, then it is used and groupIds and derivedFilterPath are ignored	
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. • This is a filter that allows the user to only generate for a specific group or list of groups found in the ConfigureStartingFolders. • Pass in null to select all groupIds.	LONGVARCHAR

Direction	Parameter Name	Parameter Type
OUT	**result TypeDefinitions.generateViewsRow** (datasourceName VARCHAR, - related data source name projectFolderName VARCHAR, - related project name greatGrandParentName VARCHAR, - lineage to data source name grandParentName VARCHAR, - lineage to catalog name parentName VARCHAR, - lineage to schema name containerName VARCHAR, - lineage to table name. containerType VARCHAR, - container resource type (i.e. TABLE) columnName VARCHAR, - physical column name. logicalColumnName VARCHAR, - target logical column name. logicalColumnType VARCHAR, - target logical column type. logicalStatus VARCHAR, - the status for the usage of this column on input, when generateMode = 'G' possible values are: • GENERATED – generated the column name based on rules supplied–UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • on input, when generateMode = 'R' possible values are: • FOUND – found when it finds a match to physical name in the spreadsheet • UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • DROPPED – gets dropped from the list when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=0 • SKIPPED – when overwrite=1, the resources is SKIPPED if it exists. containerPath TypeDefinitions.pathType – the full path to the container resource. duration HOUR TO SECOND – the time it takes to process a view or column. The time is incremental for each row in the view. The first row is the table/procedure which incurs time for initial checks. The last row (column) in the view is the total time it took to process the view. Each row is processed as it is read from the repository.)	CURSOR

View Generation: Generate Cast Views

16. **generateCastViews –** This procedure is used for generating views with cast statements. Casting can be done at any layer within the best practices. This procedure allows the user to specify the source folder or resource and target folder where the resource should be generated. It does not rely on the ConfigureStartingFolders.

    Combinations allowed:

    sourceResource → targetResource

    CONTAINER → CONTAINER [generate casting views from the contents of a CONTAINER (folder) to another CONTAINER retaining any sub-folders during the generation.]

    RESOURCE → CONTAINER [generate casting views for a resource to a target folder]

    Combinations allowed:

    RESOURCE → RESOURCE [will evaluate for phase 2]

    CONTAINER → RESOURCE [this scenario is never allowed]

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.	BIT
IN	**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	INTEGER

Direction	Parameter Name	Parameter Type
	continue processing • 2 (default)="OVERWRITE_IF_EXISTS"=overwrite the resource if it exists.	
IN	**copyAnnotation** - allows user to decide whether they want to copy annotations or not form both resource and columns. • 0 or null (default)=false=do not copy the annotation from the target resource • 1=true=do copy the annotation from the target resource	BIT
IN	**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.	BIT
IN	**exactMatch** – specified how the source resource will be matched against the target resource 0 or null (default) =fuzzy match – sourcePath + derivedFilterPath must simply be contained within resourcePath 1=exact match – sourcePath + derivedFilterPath must match exactly in resourcePath	SMALLINT
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	LONGVARCHAR
IN	**sourceResource** – the source resource may be a starting folder or a specific resource such as a view, procedure, data source, data source schema or data source table type = CONTAINER type = TABLE, PROCEDURE, DATA_SOURCE, when the type is not a container, it will extract the resource name into the	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	derivedFilterPath and use the remaining path to evaluate if it is a CONTAINER. A container must be passed into the underlying "generateViewsLoop" which does the processing. Depending on how the parameter "exactMatch" is set will determine what views get generated.	
IN	**targetResource** – the target resource may only be a folder (phase 1). In phase 2, I will evaluate the possibility of specifying a specific resource where the name may be different than the target.	LONGVARCHAR
OUT	**result TypeDefinitions.generateViewsRow** (datasourceName VARCHAR, - related data source name projectFolderName VARCHAR, - related project name greatGrandParentName VARCHAR, - lineage to data source name grandParentName VARCHAR, - lineage to catalog name parentName VARCHAR, - lineage to schema name containerName VARCHAR, - lineage to table name. containerType VARCHAR, - container resource type (i.e. TABLE) columnName VARCHAR, - physical column name. logicalColumnName VARCHAR, - target logical column name. logicalColumnType VARCHAR, - target logical column type. logicalStatus VARCHAR, - the status for the usage of this column on input, when generateMode = 'G' possible values are: • GENERATED – generated the column name based on rules supplied–UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • on input, when generateMode = 'R' possible values are: • FOUND – found when it finds a match to physical name in the spreadsheet • UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • DROPPED – gets dropped from the list when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=0 • SKIPPED – when overwrite=1, the resources is SKIPPED if it exists.	

Direction	Parameter Name	Parameter Type
	containerPath TypeDefinitions.pathType – the full path to the container resource. duration HOUR TO SECOND – the time it takes to process a view or column. The time is incremental for each row in the view. The first row is the table/procedure which incurs time for initial checks. The last row (column) in the view is the total time it took to process the view. Each row is processed as it is read from the repository.)	

View Generation: Generate Client Published Views

17. **generateClientPublished –** This procedure is used for generating "Client Published Views" either from other views or from data sources. According to the Best Practices, Client Published are created from the Application Layer, Client Views or Client Services. The objective of the Client Published sub-layer is to provide a contract with the consumer. A contract is established by creating explicit cast statements for the published views. Many BI tools will import the published data services from Composite and expect that the metadata will not change in Composite. However the danger in not explicitly casting the types in views is that a view will inherit data types from underlying views. When developers change the types in the underlying views, it will bubble up to the published views, thus changing the metadata to the published data services. This is in turn may cause run-time errors for the BI tool that is accessing Composite. The solution is to provide a published views layer that is a one-to-one correlation to a view in the client views or client services layer. The sole purpose of the view is to explicitly cast the types from the underlying view at a point in time. The names do not change.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.	BIT

Direction	Parameter Name	Parameter Type
IN	**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"=overwrite the resource if it exists.	INTEGER
IN	**copyAnnotation** - allows user to decide whether they want to copy annotations or not form both resource and columns. • 0 or null (default)=false=do not copy the annotation from the target resource • 1=true=do copy the annotation from the target resource	BIT
IN	**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.	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the “sourceResource+derivedFilterPath” is contained within the resourcePath. In this case “/shared/project/Physical/Metadata/ds_orders1/order” is contained within example paths whereby one path	BIT

Direction	Parameter Name	Parameter Type
	ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders + the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. Option 1	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. Option 2 The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is	

Direction	Parameter Name	Parameter Type
	ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ds_orders1 • derivedFilterPath="customers,orders" The result for the above is that customers and orders are the only views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	LONGVARCHAR
OPTION 1	**Explicit Folders:** Use explicit sourceResource and publishToFolder parameters with derivedFilterPath being optional.	
IN	**sourceResource –** The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder. If this is set it supercedes layerType, inGroupIDs and derivedFilterPath	
IN	**publishToFolder –**This is the full path to the folder in which to generate the views. This is only required if option 1: targetResource is provided. If sourceResource is not blank, then it is used and groupIds and derivedFilterPath are ignored	
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. Pass in null to select all groupIds.	LONGVARCHAR
OUT	**result TypeDefinitions.generateViewsRow** (datasourceName VARCHAR, - related data source name projectFolderName VARCHAR, - related project name greatGrandParentName VARCHAR, - lineage to data source name	CURSOR

Direction	Parameter Name	Parameter Type
	grandParentName VARCHAR, - lineage to catalog name parentName VARCHAR, - lineage to schema name containerName VARCHAR, - lineage to table name. containerType VARCHAR, - container resource type (i.e. TABLE) columnName VARCHAR, - physical column name. logicalColumnName VARCHAR, - target logical column name. logicalColumnType VARCHAR, - target logical column type. logicalStatus VARCHAR, - the status for the usage of this column on input, when generateMode = 'G' possible values are: • GENERATED – generated the column name based on rules supplied–UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • on input, when generateMode = 'R' possible values are: • FOUND – found when it finds a match to physical name in the spreadsheet • UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • DROPPED – gets dropped from the list when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=0 • SKIPPED – when overwrite=1, the resources is SKIPPED if it exists. containerPath TypeDefinitions.pathType – the full path to the container resource. duration HOUR TO SECOND – the time it takes to process a view or column. The time is incremental for each row in the view. The first row is the table/procedure which incurs time for initial checks. The last row (column) in the view is the total time it took to process the view. Each row is processed as it is read from the repository.)	

View Generation: Generate Application (Client) Views

18. **generateClientViews –** This procedure is used for generating "Client Views" either from other views or from data sources. According to the Best Practices, Client Views are created from the Business Layer, Logical Views.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.	BIT
IN	**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"=overwrite the resource if it exists.	INTEGER
IN	**copyAnnotation** - allows user to decide whether they want to copy annotations or not form both resource and columns. • 0 or null (default)=false=do not copy the annotation from the target resource • 1=true=do copy the annotation from the target resource	BIT
IN	**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.	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must	BIT

Direction	Parameter Name	Parameter Type
	simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the "sourceResource+derivedFilterPath" is contained within the resourcePath. In this case "/shared/project/Physical/Metadata/ds_orders1/order" is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders + the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. <u>Option 1</u> In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. <u>Option 2</u> The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. <u>Example</u>: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all	

Direction	Parameter Name	Parameter Type
	source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath=”customers,orders”,orders The result for the above is that the “customers,orders” filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ds_orders1 • derivedFilterPath=“customers,orders” The result for the above is that customers and orders are the only views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	LONGVARCHAR
OPTION 1	**Explicit Folders:** Use explicit sourceResource and publishToFolder parameters with derivedFilterPath being optional.	
IN	**sourceResource –** The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder. If this is set it supercedes layerType, inGroupIDs and derivedFilterPath	

Direction	Parameter Name	Parameter Type
IN	**publishToFolder –**This is the full path to the folder in which to generate the views. This is only required if option 1: targetResource is provided. If sourceResource is not blank, then it is used and groupIds and derivedFilterPath are ignored	
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. Pass in null to select all groupIds.	LONGVARCHAR
OUT	**result TypeDefinitions.generateViewsRow** (datasourceName VARCHAR, - related data source name projectFolderName VARCHAR, - related project name greatGrandParentName VARCHAR, - lineage to data source name grandParentName VARCHAR, - lineage to catalog name parentName VARCHAR, - lineage to schema name containerName VARCHAR, - lineage to table name. containerType VARCHAR, - container resource type (i.e. TABLE) columnName VARCHAR, - physical column name. logicalColumnName VARCHAR, - target logical column name. logicalColumnType VARCHAR, - target logical column type. logicalStatus VARCHAR, - the status for the usage of this column on input, when generateMode = 'G' possible values are: • GENERATED – generated the column name based on rules supplied–UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • on input, when generateMode = 'R' possible values are: • FOUND – found when it finds a match to physical name in the spreadsheet • UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • DROPPED – gets dropped from the list when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=0 • SKIPPED – when overwrite=1, the resources is SKIPPED if it exists.	CURSOR

Direction	Parameter Name	Parameter Type
	containerPath TypeDefinitions.pathType – the full path to the container resource. duration HOUR TO SECOND – the time it takes to process a view or column. The time is incremental for each row in the view. The first row is the table/procedure which incurs time for initial checks. The last row (column) in the view is the total time it took to process the view. Each row is processed as it is read from the repository.)	

View Generation: Generate Business (Business) Views

19. **generateBusinessViews –** This procedure is used for generating "Business Views" from Logical views. According to the Best Practices, Business Views are created from the Business layer, Business views.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.	BIT
IN	**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"=overwrite the resource if it exists.	INTEGER
IN	**copyAnnotation** - allows user to decide whether they want to copy annotations or not form both resource and columns.	BIT

Direction	Parameter Name	Parameter Type
	• 0 or null (default)=false=do not copy the annotation from the target resource • 1=true=do copy the annotation from the target resource	
IN	**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.	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the “sourceResource+derivedFilterPath” is contained within the resourcePath. In this case “/shared/project/Physical/Metadata/ds_orders1/order” is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=	BIT

Direction	Parameter Name	Parameter Type
	/shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders + the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. <u>Option 1</u> In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. <u>Option 2</u> The layerType and the groupId are used to filter the rows from the	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ ds_orders1 • derivedFilterPath="customers,orders" The result for the above is that customers and orders are the only views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	
OPTION 1	**Explicit Folders:** Use explicit sourceResource and publishToFolder parameters with derivedFilterPath being optional.	
IN	**sourceResource –** The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder. If this is set it supercedes layerType, inGroupIDs and derivedFilterPath	
IN	**publishToFolder –**This is the full path to the folder in which to generate the views. This is only required if option 1: targetResource is provided. If sourceResource is not blank, then it is used and groupIds and derivedFilterPath are ignored	
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. Pass in null to select all groupIds.	LONGVARCHAR
OUT	**result TypeDefinitions.generateViewsRow** (datasourceName VARCHAR, - related data source name projectFolderName VARCHAR, - related project name greatGrandParentName VARCHAR, - lineage to data source name grandParentName VARCHAR, - lineage to catalog name parentName VARCHAR, - lineage to schema name containerName VARCHAR, - lineage to table name. containerType VARCHAR, - container resource type (i.e. TABLE) columnName VARCHAR, - physical column name. logicalColumnName VARCHAR, - target logical column name. logicalColumnType VARCHAR, - target logical column type. logicalStatus VARCHAR, - the status for the usage of this column on input, when generateMode = 'G' possible values are:	CURSOR

Direction	Parameter Name	Parameter Type
	• GENERATED – generated the column name based on rules supplied–UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • on input, when generateMode = 'R' possible values are: • FOUND – found when it finds a match to physical name in the spreadsheet • UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • DROPPED – gets dropped from the list when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=0 • SKIPPED – when overwrite=1, the resources is SKIPPED if it exists. containerPath TypeDefinitions.pathType – the full path to the container resource. duration HOUR TO SECOND – the time it takes to process a view or column. The time is incremental for each row in the view. The first row is the table/procedure which incurs time for initial checks. The last row (column) in the view is the total time it took to process the view. Each row is processed as it is read from the repository.)	

View Generation: Generate Business (Logical) Views

20. **generateLogicalViews –** This procedure is used for generating "Logical Views" either from other views or from data sources. According to the Best Practices, Logical Views are created from the Formatting Layer, Formatting Views.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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	BIT

Direction	Parameter Name	Parameter Type
	• 1 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.	
IN	**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"=overwrite the resource if it exists.	INTEGER
IN	**copyAnnotation** - allows user to decide whether they want to copy annotations or not form both resource and columns. • 0 or null (default)=false=do not copy the annotation from the target resource • 1=true=do copy the annotation from the target resource	BIT
IN	**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.	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order	BIT

Direction	Parameter Name	Parameter Type
	Since fuzzy match is being used all resource paths are selected where the "sourceResource+derivedFilterPath" is contained within the resourcePath. In this case "/shared/project/Physical/Metadata/ds_orders1/order" is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders +	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. Option 1 In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. Option 2 The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will	

Direction	Parameter Name	Parameter Type
	be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ds_orders1 • derivedFilterPath="customers,orders" The result for the above is that customers and orders are the only views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	LONGVARCHAR
OPTION 1	**Explicit Folders:** Use explicit sourceResource and publishToFolder parameters with derivedFilterPath being optional.	
IN	**sourceResource –** The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder. If this is set it supercedes layerType, inGroupIDs and derivedFilterPath	
IN	**publishToFolder –**This is the full path to the folder in which to generate the views. This is only required if option 1: targetResource is provided. If sourceResource is not blank, then it is used and groupIds and derivedFilterPath are ignored	
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. Pass in null to select all groupIds.	LONGVARCHAR

Direction	Parameter Name	Parameter Type
OUT	**result TypeDefinitions.generateViewsRow** (datasourceName VARCHAR, - related data source name projectFolderName VARCHAR, - related project name greatGrandParentName VARCHAR, - lineage to data source name grandParentName VARCHAR, - lineage to catalog name parentName VARCHAR, - lineage to schema name containerName VARCHAR, - lineage to table name. containerType VARCHAR, - container resource type (i.e. TABLE) columnName VARCHAR, - physical column name. logicalColumnName VARCHAR, - target logical column name. logicalColumnType VARCHAR, - target logical column type. logicalStatus VARCHAR, - the status for the usage of this column on input, when generateMode = 'G' possible values are: • GENERATED – generated the column name based on rules supplied–UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • on input, when generateMode = 'R' possible values are: • FOUND – found when it finds a match to physical name in the spreadsheet • UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • DROPPED – gets dropped from the list when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=0 • SKIPPED – when overwrite=1, the resources is SKIPPED if it exists. containerPath TypeDefinitions.pathType – the full path to the container resource. duration HOUR TO SECOND – the time it takes to process a view or column. The time is incremental for each row in the view. The first row is the table/procedure which incurs time for initial checks. The last row (column) in the view is the total time it took to process the view. Each row is processed as it is read from the repository.)	CURSOR

View Generation: Generate Physical (Formatting) Views

21. **generateFormattingViews –** This procedure is used for generating "Formatting Views" either from other views or from data sources. According to the Best Practices, Format Views are created from the Physical Layer, Metadata tables or Formatting Transformation layer. Formatting views map one-to-one with the physical layer data source tables. The purpose of this is to map the physical tables into the corresponding canonical views so that all queries that are performed in the layers above the Formatting View layer are done so using the logical/canonical model.

    generateWithSourceColumn – This is an internal variable that is useful if you don't want to generate physical columns or views when the physical names are absent from the Common Model Data Dictionary spreadsheet.

    This variable is relevant when generateMode='R'

    1=Generate the view with the source column (pass through)-logical status is UNCHANGED

    0=Do NOT generate the view with the source column (no pass through)-logical status is DROPPED

    This is useful when you don't want to generate certain physical metadata columns into the Formatting layer.

    Therefore, if the physical has no definition in the Common_Model_v3_file[1-3].xls spreadsheet and this parameter is set to 0, then do not generate the logical or physical source column. The view gets generated without that columnn.

    If all of the columns including the view name have a status of DROPPED (indicating they are not in the spreadsheet), then the view is not generated at all.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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	BIT

Direction	Parameter Name	Parameter Type
	• 1 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.	
IN	**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"=overwrite the resource if it exists.	INTEGER
IN	**copyAnnotation** - allows user to decide whether they want to copy annotations or not form both resource and columns. • 0 or null (default)=false=do not copy the annotation from the target resource • 1=true=do copy the annotation from the target resource	BIT
IN	**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.	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order	BIT

Direction	Parameter Name	Parameter Type
	Since fuzzy match is being used all resource paths are selected where the "sourceResource+derivedFilterPath" is contained within the resourcePath. In this case "/shared/project/Physical/Metadata/ds_orders1/order" is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders +	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. Option 1 In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. Option 2 The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will	

Direction	Parameter Name	Parameter Type
	be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ds_orders1 • derivedFilterPath="customers,orders" The result for the above is that customers and orders are the only views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	LONGVARCHAR
OPTION 1	**Explicit Folders:** Use explicit sourceResource and publishToFolder parameters with derivedFilterPath being optional.	
IN	**sourceResource –** The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder. If this is set it supercedes layerType, inGroupIDs and derivedFilterPath	
IN	**publishToFolder –**This is the full path to the folder in which to generate the views. This is only required if option 1: targetResource is provided. If sourceResource is not blank, then it is used and groupIds and derivedFilterPath are ignored	
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. Pass in null to select all groupIds.	LONGVARCHAR

Direction	Parameter Name	Parameter Type
OUT	**result TypeDefinitions.generateViewsRow** (datasourceName VARCHAR, - related data source name projectFolderName VARCHAR, - related project name greatGrandParentName VARCHAR, - lineage to data source name grandParentName VARCHAR, - lineage to catalog name parentName VARCHAR, - lineage to schema name containerName VARCHAR, - lineage to table name. containerType VARCHAR, - container resource type (i.e. TABLE) columnName VARCHAR, - physical column name. logicalColumnName VARCHAR, - target logical column name. logicalColumnType VARCHAR, - target logical column type. logicalStatus VARCHAR, - the status for the usage of this column on input, when generateMode = 'G' possible values are: • GENERATED – generated the column name based on rules supplied–UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • on input, when generateMode = 'R' possible values are: • FOUND – found when it finds a match to physical name in the spreadsheet • UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • DROPPED – gets dropped from the list when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=0 • SKIPPED – when overwrite=1, the resources is SKIPPED if it exists. containerPath TypeDefinitions.pathType – the full path to the container resource. duration HOUR TO SECOND – the time it takes to process a view or column. The time is incremental for each row in the view. The first row is the table/procedure which incurs time for initial checks. The last row (column) in the view is the total time it took to process the view. Each row is processed as it is read from the repository.)	CURSOR

View Generation: Generate Physical (Physical) Views

22. **generatePhysicalViews –** This procedure is used for generating "Physical Views" from a data sources. This procedure has been **deprecated** in this release for the purposes of generating the one-to-one between the physical metadata and physical layer. This functionality has been subsumed by the Formatting sub-layer. However, this procedure is maintained for backward compatibility. Additionally, it may still be useful to user generatePhysicalViews() to create a set of one-to-one views used for CRUD generation. The generatCRUDOperations() procedure requires a set of views that map one-to-one with the physical metadata and do not contain any new or derived columns.

    Previous Definition: According to the Best Practices, Physical Views are created from the Physical Layer, Physical Metadata. The "Physical" map one-to-one with the "Metadata" layer. The purpose of this is to map the data source metadata into a set of views that represent a one-to-one physical abstraction layer. This allows the administrators to import a Composite ".CAR" package and automatically perform a rebind of the Physical Views to different Physical Metadata without changing any code. This is typically done at the time of moving between DEV, UAT and Production. Additionally, caching policies may be set up against the Physical Views instead of the Physical Metadata layer allowing for the definition in one place.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.	BIT
IN	**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"=overwrite the	INTEGER

Direction	Parameter Name	Parameter Type
	resource if it exists.	
IN	**copyAnnotation** - allows user to decide whether they want to copy annotations or not form both resource and columns. • 0 or null (default)=false=do not copy the annotation from the target resource • 1=true=do copy the annotation from the target resource	BIT
IN	**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.	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the “sourceResource+derivedFilterPath” is contained within the resourcePath. In this case “/shared/project/Physical/Metadata/ds_orders1/order” is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders	BIT

Direction	Parameter Name	Parameter Type
	/shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders + the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. <u>Option 1</u> In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	the derivedFilterPath. Option 2 The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ ds_orders1 • derivedFilterPath="customers,orders" The result for the above is that customers and orders are the only	

Direction	Parameter Name	Parameter Type
	views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	LONGVARCHAR
OPTION 1	**Explicit Folders:** Use explicit sourceResource and publishToFolder parameters with derivedFilterPath being optional.	
IN	**sourceResource –** The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder. If this is set it supercedes layerType, inGroupIDs and derivedFilterPath	
IN	**publishToFolder –**This is the full path to the folder in which to generate the views. This is only required if option 1: targetResource is provided. If sourceResource is not blank, then it is used and groupIds and derivedFilterPath are ignored	
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. Pass in null to select all groupIds.	LONGVARCHAR
OUT	**result TypeDefinitions.generateViewsRow** (datasourceName VARCHAR, - related data source name projectFolderName VARCHAR, - related project name greatGrandParentName VARCHAR, - lineage to data source name grandParentName VARCHAR, - lineage to catalog name parentName VARCHAR, - lineage to schema name containerName VARCHAR, - lineage to table name. containerType VARCHAR, - container resource type (i.e. TABLE) columnName VARCHAR, - physical column name. logicalColumnName VARCHAR, - target logical column name.	CURSOR

Direction	Parameter Name	Parameter Type
	logicalColumnType VARCHAR, - target logical column type. logicalStatus VARCHAR, - the status for the usage of this column on input, when generateMode = 'G' possible values are: • GENERATED – generated the column name based on rules supplied–UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • on input, when generateMode = 'R' possible values are: • FOUND – found when it finds a match to physical name in the spreadsheet • UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • DROPPED – gets dropped from the list when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=0 • SKIPPED – when overwrite=1, the resources is SKIPPED if it exists. containerPath TypeDefinitions.pathType – the full path to the container resource. duration HOUR TO SECOND – the time it takes to process a view or column. The time is incremental for each row in the view. The first row is the table/procedure which incurs time for initial checks. The last row (column) in the view is the total time it took to process the view. Each row is processed as it is read from the repository.)	

View Generation: Generate Views – generic API

23. **generateViews –** This procedure like an API call and is used for generating any layer of "Views" either from other views or from data sources. According to the Best Practices, there are 4 layers (Physical, Formatting, Busines and Mapping) each of which has one or more sub-layers. This is a general view generation procedure that must have all aspects of it configured properly to achieve the desired results. The other 4 procedures (generatePhysicalViews, generateFormattingViews, generateLogicalViews, generateClientViews) are more canned and specific to the concepts of a particular layer. Use those as guidelines for configure the parameters that get passed into this procedure.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	

Direction	Parameter Name	Parameter Type
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.	BIT
IN	**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"=overwrite the resource if it exists.	INTEGER
IN	**copyAnnotation** - allows user to decide whether they want to copy annotations or not form both resource and columns. • 0 or null (default)=false=do not copy the annotation from the target resource • 1=true=do copy the annotation from the target resource	BIT
IN	**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.	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath	BIT

Direction	Parameter Name	Parameter Type
	Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the "sourceResource+derivedFilterPath" is contained within the resourcePath. In this case "/shared/project/Physical/Metadata/ds_orders1/order" is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath.	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders + the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. Option 1 In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. Option 2 The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the	

Direction	Parameter Name	Parameter Type
	designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ds_orders1 • derivedFilterPath="customers,orders" The result for the above is that customers and orders are the only views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	LONGVARCHAR
OPTION 1	**Explicit Folders:** Use explicit sourceResource and publishToFolder parameters with derivedFilterPath being optional.	
IN	**sourceResource –** The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder. If this is set it supercedes layerType, inGroupIDs and derivedFilterPath	
IN	**generateToFolder –** This is the full path to the folder in which to	

Direction	Parameter Name	Parameter Type
	generate the views. This is only required if option 1: sourceResource is provided.	
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**layerType** • PV=Physical Views – physical views • FV=Formatting Views - formatting views • BV=Business Views - business views (only single source) • LV=Logical Views - logical views (only single source) • CV=Client Views - client views (only single source) • CP=Client Published - client published views (only single source) • DB=Database link – generate a Composite database link	VARCHAR
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. • Pass in null to select all groupIds.	LONGVARCHAR
	ADDITIONAL OPTIONS	
IN	**generateMode** • G=Generate the resource names • R=Retrieve the resource name from a spreadsheet. (resources include PATH names, VIEW names AND COLUMN names)	CHAR(1)
IN	**outputMode** • A=Return All abstract columns, • U=Return ONLY Unchanged/Dropped columns that were NOT found.	CHAR(1)
IN	**generateViews** • 0=Do not generate – (browse only) print out what will happen but don't perform the generation. • 1=Do generate [DEFAULT] – Perform the VIEW Generation with a column projection. • 2= Do generate – Perform the VIEW Generation with a select * projection.	SMALLINT
IN	**resourceCaseRule** used when generateMode='G'	CHAR(1)

Direction	Parameter Name	Parameter Type
	TABLES only. This resourceCaseRule is only used for tables. Assumption: The original table name has to have separators (_) for this to work properly. If the original word has no separators then the case rule gets applied to the single word. • j=javaCase – 1st word part is lower case, following word parts are 1st is letter is upper and remaining word part is lower with no separators • C=CamelCase – 1st letter of each word part is upper case and remaining word part is lower with no separators • T=Title_Case – 1st letter of each word part is upper case and remaining word part is lower with separators retained • U=UPPER_CASE – All word parts are UPPER case with 103eparators retained • l=lower_case – All word parts are lower case with separators retained • O=(default) Original_case – The word is not changed at all – just pass it through as is	
IN	**columnCaseRule** used when generateMode='G' COLUMNS only. This columnCaseRule is only used for tables. Assumption: The original column name has to have separators (_) for this to work properly. If the original word has no separators then the case rule gets applied to the single word. • j=javaCase – 1st word part is lower case, following word parts are 1st is letter is upper and remaining word part is lower with no separators • C=CamelCase – 1st letter of each word part is upper case and remaining word part is lower with no separators • T=Title_Case – 1st letter of each word part is upper case and remaining word part is lower with separators retained • U=UPPER_CASE – All word parts are UPPER case with 103eparators retained • l=lower_case – All word parts are lower case with separators retained • O=(default) Original_case – The word is not changed at all – just pass it through as is	CHAR(1)
IN	**useAliasRule**	SMALLINT

Direction	Parameter Name	Parameter Type
	Used when generateMode='G' • 0=(default) DO NOT perform alias rule lookup at all. Word Part is passed through • 1=Use alias rule and MATCH CASE exactly • 2=Use alias rule and DO NOT MATCH CASE	
IN	**resourcePrefix** Used when generateMode='G' Any set of characters used to prefix a table or procedure name. Include underscores with the suffix if applicable. Example: V_ MY_TABLE or V_MY_TABLE	VARCHAR
IN	**resourceSuffix** Used when generateMode='G' Any set of characters used to suffix a table or procedure name. Include underscores with the suffix if applicable. Example: _APP MY_TABLE or MY_TABLE_APP	VARCHAR
IN	**newColumnList** Used when generateMode='G' A formatted list of new columns to add to the end of the view. The column will not be added if it already exists. The format is as follows: column1&&type1&&value1//column2&&type2&&value2	LONGVARCHAR
IN	**generateWithSourceColumn** • 1=(default) Generate the view with the source column (pass through)-Column status is UNCHANGED • 0=Do NOT generate the view with the source column (no pass through)-Column status is DROPPED	SMALLINT
IN	**generateCast** • 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. • 0=Do not generate CAST statement. Pass through column as is. Default behavior. • 1=Generate the CAST statement around the column • 2-Generate the CAST statement around the non-index columns only (No CAST on index columns)	SMALLINT

Direction	Parameter Name	Parameter Type
	• 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) • 4-Generate the CAST statement around the non-index columns and non-primary key index columns only (No CAST on primary key index columns) • 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)	
IN	**generateIndexes** • This parameter allows the user to control whether to generate indexes on the target views as derived from the underlying resource. • 1=(default) Generate indexes. • 0=Do not generate indexes.	SMALLINT
IN	**generateUnsupportedColumnType** • This flag indicates whether to ignore or generate unsupported column types. For example in Oracle an SDO spatial type gets imported into Composite as 'OTHER'. • 0/null (default) - ignore column type = 'OTHER' and do not generate that column • 1 - generate columns where the column type = 'OTHER'	BIT
OUT	**result TypeDefinitions.generateViewsRow** (datasourceName VARCHAR, - related data source name projectFolderName VARCHAR, - related project name greatGrandParentName VARCHAR, - lineage to data source name grandParentName VARCHAR, - lineage to catalog name parentName VARCHAR, - lineage to schema name containerName VARCHAR, - lineage to table name. containerType VARCHAR, - container resource type (i.e. TABLE) columnName VARCHAR, - physical column name. logicalColumnName VARCHAR, - target logical column name. logicalColumnType VARCHAR, - target logical column type. logicalStatus VARCHAR, - the status for the usage of this column on input, when generateMode = 'G' possible values are: • GENERATED – generated the column name based on	CURSOR = TypeDefinitions.ComonodelV2Row

Direction	Parameter Name	Parameter Type
	rules supplied–UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • on input, when generateMode = 'R' possible values are: • FOUND – found when it finds a match to physical name in the spreadsheet • UNCHANGED – remains unchanged when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=1 • DROPPED – gets dropped from the list when it cannot match to the physical resourceName and the ConfigureParams.generateWithSourceColumn=0 • SKIPPED – when overwrite=1, the resources is SKIPPED if it exists. containerPath TypeDefinitions.pathType – the full path to the container resource. duration HOUR TO SECOND – the time it takes to process a view or column. The time is incremental for each row in the view. The first row is the table/procedure which incurs time for initial checks. The last row (column) in the view is the total time it took to process the view. Each row is processed as it is read from the repository.)	

Column Generation: Generate Datasource List

These scripts are used for generating the column definitions for views.

24. **generateDatasourceList –** This procedure is used for generating a list of columns either from a folder of views or from data sources. This can be very useful when you need to understand what all the columns are in your data source. It is also possible to generate the logical names using various case sensitivity rules and alias rules. This could be done if you do not already have a data dictionary defined for your physical to logical mappings. This is one way to begin a data dictionary for the physical to logical mapping. See "generateDatasourceListCSV" for the procedure to write the results to a file.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateLogicalNames** • 0 (FALSE) – generate datasource list only with no logical metadata (names, types, transformation, definition) • 1 (TRUE) – generate datasource list with logical metadata	BIT

Direction	Parameter Name	Parameter Type
	(names, types, transformations, definition) Generally this parameter is true. However, it may be useful to set to generateLogicalNames=0 and layerType='PM' which allows the user to generate just the physical names for the physical metadata in the spreadsheet format as a way of initializing the physical names only.	
IN	**generateMode** Determines whether to retrieve names from the Common_Model_v3_file[1-3].xlsx spreadsheets or generate them and is only meaningful when generateLogicalNames=1 • G=Generate the resource names • R=Retrieve the resource name from a spreadsheet. (resources include PATH names, VIEW names AND COLUMN names) • Use “R” when performing the round-trip between the formatting layer and the spreadsheet.	VARCHAR
IN	**resourceCaseRule** used when generateMode='G' TABLES only. This resourceCaseRule is only used for tables. Assumption: The original table name has to have separators (_) for this to work properly. If the original word has no separators then the case rule gets applied to the single word. • j=javaCase – 1st word part is lower case, following word parts are 1st is letter is upper and remaining word part is lower with no separators • C=CamelCase – 1st letter of each word part is upper case and remaining word part is lower with no separators • T=Title_Case – 1st letter of each word part is upper case and remaining word part is lower with separators retained • U=UPPER_CASE – All word parts are UPPER case with 107eparators retained • l=lower_case – All word parts are lower case with separators retained O=(default) Original_case – The word is not changed at all – just pass it through as is	CHAR(1)
IN	**columnCaseRule** used when generateMode='G' COLUMNS only. This columnCaseRule is only used for tables. Assumption: The original column name has to have separators (_)	CHAR(1)

Direction	Parameter Name	Parameter Type
	for this to work properly. If the original word has no separators then the case rule gets applied to the single word. • j=javaCase – 1st word part is lower case, following word parts are 1st is letter is upper and remaining word part is lower with no separators • C=CamelCase – 1st letter of each word part is upper case and remaining word part is lower with no separators • T=Title_Case – 1st letter of each word part is upper case and remaining word part is lower with separators retained • U=UPPER_CASE – All word parts are UPPER case with 108eparators retained • l=lower_case – All word parts are lower case with separators retained O=(default) Original_case – The word is not changed at all – just pass it through as is	
IN	**useAliasRule** Used when generateMode='G' and determines how to use the word part alias rules. • 0=[default] DO NOT perform alias rule lookup at all. Word Part is passed through (default is 0) • 1=Use alias rule and MATCH CASE exactly • 2=Use alias rule and DO NOT MATCH CASE • The "AliasNameRuleSet()" procedure is found in the folder /shared/<project-folder>/_scripts/Configure.	SMALLINT
IN	**resourcePrefix** Used when generateMode='G' Any set of characters used to prefix a table or procedure name. Include underscores with the suffix if applicable. Example: V_ MY_TABLE or V_MY_TABLE	VARCHAR
IN	**resourceSuffix** Used when generateMode='G' Any set of characters used to suffix a table or procedure name. Include underscores with the suffix if applicable. Example: _APP MY_TABLE or MY_TABLE_APP	VARCHAR
IN	**newColumnList** Used when generateMode='G' A formatted list of new columns to add to the end of the view. The	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	column will not be added if it already exists. The format is as follows: column1&&type1&&value1//column2&&type2&&value2	
IN	**generateUnsupportedColumnType** • This flag indicates whether to ignore or generate unsupported column types. For example in Oracle an SDO spatial type gets imported into Composite as 'OTHER'. • 0/null (default) - ignore column type = 'OTHER' and do not generate that column • 1 - generate columns where the column type = 'OTHER'	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the "sourceResource+derivedFilterPath" is contained within the resourcePath. In this case "/shared/project/Physical/Metadata/ds_orders1/order" is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders	BIT

Direction	Parameter Name	Parameter Type
	In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders + the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. <u>Option 1</u> In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. <u>Option 2</u> The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ ds_orders1 • derivedFilterPath="customers,orders" The result for the above is that customers and orders are the only views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	
OPTION 1	**Explicit Folders:** Use explicit sourceResource parameter with derivedFilterPath being optional.	
IN	**sourceResource** • The source folder in CIS to begin searching for views to generate the view list or a source view (exact path) to generate for. • If this is set it supercedes layerType, groupIds and derivedFilterPath	LONGVARCHAR
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**layerType** • PM=Physical Metadata - physical metadata tables • FV=Formatting Views - formatting views • BV=Business Views - business views (only single source) • LV=Logical Views - logical views (only single source) • CV=Client Views - client views (only single source) • CP=Client Published - client published views (only single source)	VARCHAR
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. • Pass in null to select all groupIds.	LONGVARCHAR
OUT	**result** CURSOR TypeDefinitions.CommonModelV2Row (DataSource VARCHAR(255), -- The name of the data source in composite (data lineage-used resources) ProjectFolderName VARCHAR(255), -- The project folder name is the last name found in the defaultValues.basePath for a project. GreatGrandParentName VARCHAR(255), -- The name of the great grand parent container (data source name) GrandParentName VARCHAR(255), -- The name of the grand parent container (catalog name) ParentName VARCHAR(255), -- The name of the parent container -	CURSOR

Direction	Parameter Name	Parameter Type
	parent to the parent (a.k.a. grandparent) ContainerName VARCHAR(255), -- The name of the container - parent to the resource PhysicalName VARCHAR(255), -- The physical column name (a.k.a. source column name) PhysicalType VARCHAR(255), -- The physical type (a.k.a. source column native type) LogicalName VARCHAR(255), -- The logical column name (a.k.a. table/view alias) LogicalType VARCHAR(255), -- The logical column type LogicalTransformation LONGVARCHAR, -- The logical column transformation excluding outer cast statement LogicalDefinition LONGVARCHAR, -- The logical resource definition (a.k.a. view/table/column annotation) LogicalPath LONGVARCHAR -- The logical Path (this is not dumped to the spreadsheet) Duration INTERVAL HOUR TO SECOND -- The time it takes to process a view)	

Column Generation: Generate Datasource List to CSV file

These scripts are used for generating the column definitions for views to a CSV file.

25. **generateDatasourceListCSV –** This procedure is used for generating the Best Practices spreadsheet as a CSV file. The columns that are output to the spreadsheet match exactly with the format for Columns A-L in Common_Model_v1_file[1-3].xlsx.

    This can be very useful when you need to understand what all the columns are in your data source. It is also possible to generate the logical names using various case sensitivity rules and alias rules. This could be done if you do not already have a data dictionary defined for your physical to logical mappings. This is one way to begin a data dictionary for the physical to logical mapping.

    25.1. **Use case:** Generate Best Practices Spreadsheet from Formatting Views

    Perform this step when upgrading Best Practices or when the need to synchronize the spreadsheet with the actual views is required.

    This is useful in that the user can perform a “loss-less” round-trip between the actual views in the Formatting sub-layer and the Common_Model_v1_file[1-3].xlsx.

Option 1: Use “targetResource” to specify a specific folder

targetResource=<your path to Formatting >

generateMode=R

generateLogicalNames=1

layerType=null

groupIds=null

derivedFilterPath=null

Option 2: Use “groupIds” for ConfigureStartingFolders

targetResource=null

generateMode=R

generateLogicalNames=1

layerType=FV

groupIds=<your list of group ids>

derivedFilterPath=<optional>

25.2. **Use case:** Generate Physical Metadata Columns with Logical Names/Types

Perform this step when you don’t have any spreadsheet to start with. This can be used to populate an initial spreadsheet which can then be modified according the “Physical to Logical” mappings.

It is also possible to generate the logical names using various case sensitivity rules and alias rules. This could be done if you do not already have a data dictionary defined for your physical to logical mappings. This is one way to begin a data dictionary for the physical to logical mapping.

Option 1: Use “targetResource” to specify a specific folder

targetResource=<your path to Metadata>

generateMode=G

caseRule=O [Determine what case the logical views should have.]

generateLogicalNames=1

layerType=null

groupIds=null

derivedFilterPath=null

Option 2: Use “groupIds” for ConfigureStartingFolders

targetResource=null

generateMode=G

caseRule=O [Determine what case the logical views should have.]

generateLogicalNames=1

layerType=PM

groupIds=<your list of group ids>

derivedFilterPath=<optional>

25.3. **Use case:** Generate Physical Metadata Columns with NO Logical Metadata

Perform this step when you don’t have any spreadsheet to start with and you simply want to generate the spreadsheet with a list of all the physical metadata and not logical metadata.

Option 1: Use “targetResource” to specify a specific folder

targetResource=<your path to Metadata>

generateMode=G

generateLogicalNames=0

layerType=null

groupIds=null

derivedFilterPath=null

Option 2: Use “groupIds” for ConfigureStartingFolders

targetResource=null

generateMode=G

generateLogicalNames=0

layerType=PM

groupIds=<your list of group ids>

derivedFilterPath=<optional>

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**csvFullPath** – full server path to write out the CSV file	LONGVARCHAR

Direction	Parameter Name	Parameter Type
IN	**bufferSize** – number of rows to buffer. Recommend 1000	INTEGER
IN	**generateHeader** – generate the header or not • 0 (FALSE) - do not generate the header • 1 (TRUE) - do generate the header	BIT
IN	**generateLogicalNames** • 0 (FALSE) – generate datasource list only with no logical metadata (names, types, transformation, definition) • 1 (TRUE) – generate datasource list with logical metadata (names, types, transformations, definition) Generally this parameter is true. However, it may be useful to set to generateLogicalNames=0 and layerType='PM' which allows the user to generate just the physical names for the physical metadata in the spreadsheet format as a way of initializing the physical names only.	BIT
IN	**generateMode** Determines whether to retrieve names from the Common_Model_v3_file[1-3].xls spreadsheets or generate them and is only meaninfful when generateLogicalNames=1 • G=Generate the resource names • R=Retrieve the resource name from a spreadsheet. (resources include PATH names, VIEW names AND COLUMN names) • Use “R” when performing the round-trip between the formatting layer and the spreadsheet.	VARCHAR
IN	**resourceCaseRule** used when generateMode='G' TABLES only. This resourceCaseRule is only used for tables. Assumption: The original table name has to have separators (_) for this to work properly. If the original word has no separators then the case rule gets applied to the single word. • j=javaCase – 1st word part is lower case, following word parts are 1st is letter is upper and remaining word part is lower with no separators • C=CamelCase – 1st letter of each word part is upper case and remaining word part is lower with no separators • T=Title_Case – 1st letter of each word part is upper case and remaining word part is lower with separators retained • U=UPPER_CASE – All word parts are UPPER case with 116eparators retained	CHAR(1)

Direction	Parameter Name	Parameter Type
	• l=lower_case – All word parts are lower case with separators retained O=(default) Original_case – The word is not changed at all – just pass it through as is	
IN	**columnCaseRule** used when generateMode='G' COLUMNS only. This columnCaseRule is only used for tables. Assumption: The original column name has to have separators (_) for this to work properly. If the original word has no separators then the case rule gets applied to the single word. • j=javaCase – 1st word part is lower case, following word parts are 1st is letter is upper and remaining word part is lower with no separators • C=CamelCase – 1st letter of each word part is upper case and remaining word part is lower with no separators • T=Title_Case – 1st letter of each word part is upper case and remaining word part is lower with separators retained • U=UPPER_CASE – All word parts are UPPER case with 117eparators retained • l=lower_case – All word parts are lower case with separators retained O=(default) Original_case – The word is not changed at all – just pass it through as is	CHAR(1)
IN	**useAliasRule** Used when generateMode='G' and determines how to use the word part alias rules. • 0=[default] DO NOT perform alias rule lookup at all. Word Part is passed through (default is 0) • 1=Use alias rule and MATCH CASE exactly • 2=Use alias rule and DO NOT MATCH CASE The "AliasNameRuleSet()" procedure is found in the folder /shared/<project-folder>/_scripts/Configure.	SMALLINT
IN	**resourcePrefix** Used when generateMode='G' Any set of characters used to prefix a table or procedure name. Include underscores with the suffix if applicable. Example: V_ MY_TABLE or V_MY_TABLE	VARCHAR
IN	**resourceSuffix**	VARCHAR

Direction	Parameter Name	Parameter Type
	Used when generateMode='G' Any set of characters used to suffix a table or procedure name. Include underscores with the suffix if applicable. Example: _APP MY_TABLE or MY_TABLE_APP	
IN	**newColumnList** Used when generateMode='G' A formatted list of new columns to add to the end of the view. The column will not be added if it already exists. The format is as follows: column1&&type1&&value1//column2&&type2&&value2	LONGVARCHAR
IN	**generateUnsupportedColumnType** • This flag indicates whether to ignore or generate unsupported column types. For example in Oracle an SDO spatial type gets imported into Composite as 'OTHER'. • 0/null (default) - ignore column type = 'OTHER' and do not generate that column • 1 - generate columns where the column type = 'OTHER'	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the “sourceResource+derivedFilterPath” is contained within the resourcePath. In this case “/shared/project/Physical/Metadata/ds_orders1/order” is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact:	BIT

Direction	Parameter Name	Parameter Type
	resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders + the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. <u>Option 1</u> In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. Option 2 The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ ds_orders1 • derivedFilterPath="customers,orders"	

Direction	Parameter Name	Parameter Type
	The result for the above is that customers and orders are the only views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during • Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	LONGVARCHAR
OPTION 1	**Explicit Folders:** Use explicit sourceResource parameter with derivedFilterPath being optional.	
IN	**sourceResource** • The source folder in CIS to begin searching for views to generate the view list or a source view (exact path) to generate for. • If this is set it supercedes layerType, groupIds and derivedFilterPath	LONGVARCHAR
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**layerType** • PM=Physical Metadata - physical metadata tables • FV=Formatting Views - formatting views • BV=Business Views - business views (only single source) • LV=Logical Views - logical views (only single source) • CV=Client Views - client views (only single source) • CP=Client Published - client published views (only single source)	VARCHAR
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. • Pass in null to select all groupIds.	LONGVARCHAR
OUT	**Error**	INTEGER

Column Generation: Generate Datasource List Insert Database

These scripts are used for generating the column definitions for views and inserting them into the postgres cache database.

26. **generateDatasourceListInsertDB –** This procedure is used for generating a list of columns either from a folder of views or from data sources (Physical/Metadata). This procedure inserts the data into a table which gets used by the Best Practices Data Abstraction procedures to generate the views. The table is called common_model_v3 and is resident in the postgres cache database that comes with DV. Initially, it has to be created and the /shared/ASAssets/BestPractices_v81/DataSource/CommonModelCache data source has to be configured. Lastly, the common_model view is synchronously cached. Note: common_model is used by the various generateViews procedures when generateMode='R' which indicates it is retrieving.

    This can be very useful when you need to understand what all the columns are in your data source. It is also possible to generate the logical names using various case sensitivity rules and alias rules. This could be done if you do not already have a data dictionary defined for your physical to logical mappings. This is one way to begin a data dictionary for the physical to logical mapping. See “generateDatasourceListCSV” for the procedure to write the results to a file.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**performDeleteProjectRows** • Y=delete all project records. • N=do not delete (updates only). If “Y” then delete all rows where /shared/ASAssets/BestPractices_v81/DataSource/common_model_v3/ProjectFolderName = <project_name>	CHAR(1)
IN	**performInsertUpdate** • Y=perform SQL operation. N=do not perform SQL operation and only display results.	CHAR(1)
IN	**refreshCache** • Y=perform SQL operation. • N=do not perform SQL operation and only display results.	CHAR(1)
IN	**generateLogicalNames** • 0 (FALSE) – generate datasource list only with no logical metadata (names, types, transformation, definition) • 1 (TRUE) – generate datasource list with logical metadata (names, types, transformations, definition) Generally, this parameter is true. However, it may be useful to set	BIT

Direction	Parameter Name	Parameter Type
	to generateLogicalNames=0 and layerType='PM' which allows the user to generate just the physical names for the physical metadata in the spreadsheet format as a way of initializing the physical names only.	
IN	**generateMode** Determines whether to retrieve names from the Common_Model_v3_file[1-3].xlsx spreadsheets or generate them and is only meaningful when generateLogicalNames=1 • G=Generate the resource names • R=Retrieve the resource name from a spreadsheet. (resources include PATH names, VIEW names AND COLUMN names) • Use “R” when performing the round-trip between the formatting layer and the spreadsheet.	VARCHAR
IN	**resourceCaseRule** used when generateMode='G' TABLES only. This resourceCaseRule is only used for tables. Assumption: The original table name has to have separators (_) for this to work properly. If the original word has no separators then the case rule gets applied to the single word. • j=javaCase – 1st word part is lower case, following word parts are 1st is letter is upper and remaining word part is lower with no separators • C=CamelCase – 1st letter of each word part is upper case and remaining word part is lower with no separators • T=Title_Case – 1st letter of each word part is upper case and remaining word part is lower with separators retained • U=UPPER_CASE – All word parts are UPPER case with 123eparators retained • l=lower_case – All word parts are lower case with separators retained O=(default) Original_case – The word is not changed at all – just pass it through as is	CHAR(1)
IN	**columnCaseRule** used when generateMode='G' COLUMNS only. This columnCaseRule is only used for tables. Assumption: The original column name has to have separators (_) for this to work properly. If the original word has no separators then the case rule gets applied to the single word.	CHAR(1)

Direction	Parameter Name	Parameter Type
	• j=javaCase – 1st word part is lower case, following word parts are 1st is letter is upper and remaining word part is lower with no separators • C=CamelCase – 1st letter of each word part is upper case and remaining word part is lower with no separators • T=Title_Case – 1st letter of each word part is upper case and remaining word part is lower with separators retained • U=UPPER_CASE – All word parts are UPPER case with 124eparators retained • l=lower_case – All word parts are lower case with separators retained O=(default) Original_case – The word is not changed at all – just pass it through as is	
IN	**useAliasRule** Used when generateMode='G' and determines how to use the word part alias rules. • 0=[default] DO NOT perform alias rule lookup at all. Word Part is passed through (default is 0) • 1=Use alias rule and MATCH CASE exactly • 2=Use alias rule and DO NOT MATCH CASE • The "AliasNameRuleSet()" procedure is found in the folder /shared/<project-folder>/_scripts/Configure.	SMALLINT
IN	**resourcePrefix** Used when generateMode='G' Any set of characters used to prefix a table or procedure name. Include underscores with the suffix if applicable. Example: V_ MY_TABLE or V_MY_TABLE	VARCHAR
IN	**resourceSuffix** Used when generateMode='G' Any set of characters used to suffix a table or procedure name. Include underscores with the suffix if applicable. Example: _APP MY_TABLE or MY_TABLE_APP	VARCHAR
IN	**newColumnList** Used when generateMode='G' A formatted list of new columns to add to the end of the view. The column will not be added if it already exists. The format is as follows:	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	column1&&type1&&value1//column2&&type2&&value2	
IN	**generateUnsupportedColumnType** • This flag indicates whether to ignore or generate unsupported column types. For example in Oracle an SDO spatial type gets imported into Composite as 'OTHER'. • 0/null (default) - ignore column type = 'OTHER' and do not generate that column • 1 - generate columns where the column type = 'OTHER'	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the "sourceResource+derivedFilterPath" is contained within the resourcePath. In this case "/shared/project/Physical/Metadata/ds_orders1/order" is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only	BIT

Direction	Parameter Name	Parameter Type
	"/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders + the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. <u>Option 1</u> In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. <u>Option 2</u> The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ ds_orders1 • derivedFilterPath="customers,orders" The result for the above is that customers and orders are the only views generated.	
IN	**excludeDsPathsList** • Comma separated list of resource paths or partials paths to exclude. This may be useful when a data source has been moved and it's index or foreign keys • Are pointing to another data source that does not exist anymore. It may be necessary to exclude that path or paths. If an exception is thrown during	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	• Execution, try excluding those paths. • Values: /shared/MyPath/Physical/Metadata/MyDatasource	
OPTION 1	**Explicit Folders:** Use explicit sourceResource parameter with derivedFilterPath being optional.	
IN	**sourceResource** • The source folder in CIS to begin searching for views to generate the view list or a source view (exact path) to generate for. • If this is set it supercedes layerType, groupIds and derivedFilterPath	LONGVARCHAR
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**layerType** • PM=Physical Metadata - physical metadata tables • FV=Formatting Views - formatting views • BV=Business Views - business views (only single source) • LV=Logical Views - logical views (only single source) • CV=Client Views - client views (only single source) • CP=Client Published - client published views (only single source)	VARCHAR
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. • Pass in null to select all groupIds.	LONGVARCHAR
OUT	**result** CURSOR TypeDefinitions.CommonModelV2Row (DataSource VARCHAR(255), -- The name of the data source in composite (data lineage-used resources) ProjectFolderName VARCHAR(255), -- The project folder name is the last name found in the defaultValues.basePath for a project. GreatGrandParentName VARCHAR(255), -- The name of the great grand parent container (data source name) GrandParentName VARCHAR(255), -- The name of the grand parent container (catalog name) ParentName VARCHAR(255), -- The name of the parent container - parent to the parent (a.k.a. grandparent) ContainerName VARCHAR(255), -- The name of the container -	CURSOR

Direction	Parameter Name	Parameter Type
	parent to the resource PhysicalName VARCHAR(255), -- The physical column name (a.k.a. source column name) PhysicalType VARCHAR(255), -- The physical type (a.k.a. source column native type) LogicalName VARCHAR(255), -- The logical column name (a.k.a. table/view alias) LogicalType VARCHAR(255), -- The logical column type LogicalTransformation LONGVARCHAR, -- The logical column transformation excluding outer cast statement LogicalDefinition LONGVARCHAR, -- The logical resource definition (a.k.a. view/table/column annotation) LogicalPath LONGVARCHAR -- The logical Path (this is not dumped to the spreadsheet) Duration INTERVAL HOUR TO SECOND -- The time it takes to process a view)	

CRUD Generation: Generate CRUD Operations

These scripts are used to generate the Create, Read, Update and Delete (CRUD) procedures and type definition procedure.

27. **generateCRUDOperations –** This procedure is used for generating Read/Write CRUD operation procedures. In this context, CRUD stands for "Create", "Read", "Retrieve Primary Key", "Update", and "Delete". Those are the base operations. In addition to those base operations, there is a coordinator which is a procedure used to coordinate the lower level CRUD procedures.

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to console window. Do not print to the cursor result window.	BIT

Direction	Parameter Name	Parameter Type
	The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.	
IN	**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"=overwrite the resource if it exists.	INTEGER
IN	**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.	BIT
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the “sourceResource+derivedFilterPath” is contained within the resourcePath. In this case “/shared/project/Physical/Metadata/ds_orders1/order” is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath	BIT

Direction	Parameter Name	Parameter Type
	Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source resource path. The derivedFilterPath may be used with either "option 1" sourceResource or "option 2" layerType and groupId. Either way, a source resource path is present. The following rule is the same for both options: • The higher up the folder chain you specify in the sourceResource or ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource=/shared/project/Physical/Metadata derivedFilterPath=/ds_orders1/order • In the above example, the sourceResource is referencing the folder path at a higher level therfore the derivedFilterPath must include "the remaining" folders + the resource to be filtered on. In order to filter on just the "order" table/view, the user needs to add any reaming folders between the sourceResource and the target table/view or procedure. Option 1 In option 1, the sourceResource may specify a folder, view or procedure resource. If a folder is specified, the derivedFilterpath may also be specified. If you specify a sourceResource all the way down to an actual	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	table/view or procedure resource and not a folder resource, the derivedFilterPath will not be used. The reason is that the generate views will automatically extract the folder path for the sourceResource and place the table/procedure resource name in the derivedFilterPath. Option 2 The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. Example: layerType=DB sourceFolderPath= /shared/project/Application/Published derivedFilterPath="Orders_Open,/ds_orders1/Customers" Even though there are several other views in the /ds_orders1 folder under the Application/Published, only the ones specified in the filter path will be generated to the Composite Database. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a container/folder resource. If sourceResource points to a table/view or procedure resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ ds_orders1	

Direction	Parameter Name	Parameter Type
	• derivedFilterPath="customers,orders" The result for the above is that customers and orders are the only views generated.	
IN	**typeDefProcName –** The name of the type definition procedure. e.g. TypeDefinitions. If null, then the default name 'TypeDefinitions' is used.	VARCHAR
OPTION 1	**Explicit Folders:** Use explicit sourceResource and publishToFolder parameters with derivedFilterPath being optional.	
IN	**sourceResource –** The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder. If this is set it supercedes layerType, inGroupIDs and derivedFilterPath	
IN	**generateToFolder –**This is the full path to the folder in which to generate the views. This is only required if option 1: targetResource is provided. If sourceResource is not blank, then it is used and groupIds and derivedFilterPath are ignored	
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**layerType** • CR=CRUD Source Folder Path (CR). When using groupIds, only layerType=CR is permitted.	VARCHAR
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. • Pass in null to select all groupIds.	LONGVARCHAR
OUT	**result** CURSOR TypeDefinitions.generateCRUDRow (datasourceName VARCHAR(255), projectFolderName VARCHAR(255), greatGrandParentName VARCHAR(255), grandParentName VARCHAR(255), parentName VARCHAR(255), containerName VARCHAR(255), containerType VARCHAR(255), name VARCHAR(255), logicalName VARCHAR(255), logicalType VARCHAR(255), logicalStatus VARCHAR(255),	CURSOR

Direction	Parameter Name		Parameter Type
	containerPath	TypeDefinitions.pathType	
	duration	INTERVAL HOUR TO SECOND	
	);		

CRUD Generation Folder Contents:

CRUD = Create, Read, Update, Delete

The following details the folder structure that is generated when using the generateCRUDOperation(). CRUD operations are generated in the folder: <basePath>/Application/Services/CRUD/<groupId_path>.

If imported (BestPractices_vX_X_X_DataAbstractionSample81.car), an example can found in

/shared/DataAbstractionSample/Application/Services

/**CRUD/<groupId_path>** - Default CRUD folder + groupId path

/**Coordinate** - A coordinator (save) procedure is generated for every target view.

/**test** - A test harness procedure to invoke the Coordinator procedure.

/**Create** - A create procedure is generated for every target view.

/**Custom** - A custom procedure that allows the user to modify the values prior to create.

/**test** - A test harness procedure to invoke the Create procedure.

/**Definitions** - A type definitions procedure is generated with the public types for each view

Note: if generating CRUD procedures, "TypeDefinitionsGen()" will be automatically generated and will overwrite any existing procedure named "TypeDefinitionsGen"

/**Delete** - A delete procedure is generated for every target view.

/**test** - A test harness procedure to invoke the Delete procedure.

/**isEmpty** - A procedure used to test whether all fields in the vector are empty.

/**test** - A test harness procedure to invoke the isEmpty procedure.

/**Read** - A read procedure is generated for every target view.

/**test** - A test harness procedure to invoke the Read procedure.

/**RetrievePK** - A retrieve primary key procedure is generated for every target view.

/**test** - A test harness procedure to invoke the RetrievePK procedure.

/**Update** - An update procedure is generated for every target view.

/**Custom** - A custom procedure that allows the user to modify the values prior to update.

/**test** - A test harness procedure to invoke the Update procedure.

/**Utility** - A folder to place the custom utility procedures that may be needed for CRUD.

CRUD Generation: Generate Type Definitions

These scripts are used to generate the type definition procedure from a set of views.

28. **generateTypeDefinitions –** This procedure is used for generating "Type Definitions" for use by the Read/Write CRUD procedures. According to the Best Practices, Type Definitions should be created against the layer of views that will be used to execute Create, RetrievePK, Update and Delete operations against. Generally speaking, CRUD operations would be created against the top-level view, generally the Application Layer, Client Views. This procedure is provided as a separate executable procedure for convenience. The functionality of generating Type Definitions is automatically performed within the context of executing 'generateCRUDOperations()'..

Direction	Parameter Name	Parameter Type
	GENERAL PARAMETERS	
IN	**generateViewsWrapper** – toggle to wrap the cursor output or not. • 0 = print the output to the cursor result window. 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 or (default null) = print TABLE only information to console window. Do not print to the cursor result window. The aforementioned limits do not apply. • 2 = print TABLE and COLUMN information to console window. Do not print to the cursor result window.	BIT
IN	**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"=overwrite the resource if it exists.	INTEGER
IN	**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.	BIT

Direction	Parameter Name	Parameter Type
IN	**exactMatch –** specified how the source resource will be matched against the target resource • 0=fuzzy match - sourcePath + derivedFilterPath must simply be contained within resourcePath Example of Fuzzy: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource: /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=order Since fuzzy match is being used all resource paths are selected where the "sourceResource+derivedFilterPath" is contained within the resourcePath. In this case "/shared/project/Physical/Metadata/ds_orders1/order" is contained within both example paths whereby one path ends in /orders and the other one ends in /orderdetails. • 1 (default)=exact match - sourcePath + derivedFilterPath must match exactly in resourcePath Example of Exact: resourcePaths: /shared/project/Physical/Metadata/ds_orders1/orders /shared/project/Physical/Metadata/ds_orders1/orderdetails sourceResource= /shared/project/Physical/Metadata/ds_orders1 derivedFilterPath=orders In the above example, an exact match must be made between the resourcePath and the "sourceResource+derivedFilterPath". Therefore, only "/shared/project/Physical/Metadata/ds_orders1/orders" is selected because it exactly matches the resource path "/shared/project/Physical/Metadata/ds_orders1/orders".	BIT
IN	**derivedFilterPath** The path is derived by concatenating the partial filter path with the source path of the designated layer type. The layerType and the groupId are used to filter the rows from the ConfigureStartingFolders. The source path is defined in ConfigureStartingFolders per the layerType and groupId	LONGVARCHAR

Direction	Parameter Name	Parameter Type
	combination. Depending on what path you have defined for the source designator for a layer type in the ConfigureStartingFolders will determine how much of a path you will need to specify in the filter. The higher up the folder chain you specify in ConfigureStartingFolders, the more path you will need to provide for the derivedFilterPath. Exmple: layerType=CR sourceFolderPath=/shared/ASAssets/BestPractices_vXX/DataAbstractionSample/ Physical/Physical/ds_orders derivedFilterPath=customers,orders Even though there are several other views in the /Orders folder under the Application/Views, only the ones specified in the filter path will be generated to the Application/Published. Views directly under the source path do not require any qualifying path except the view name. Lastly, a leading '/' is not required but may be present if desired. If null, generate the views from all source folders as directed by the ConfigureStartingFolders and the designated layer type. **Pairing**: If you have multiple groupIds, you may pair up the derivedFilterPath items with commas. If you want multiple filters per groupId then place a double quote around those filters followed by a comma and another filter. For example: • groupIds=ds_orders1,ds_orders2 • derivedFilterPath="customers,orders",orders The result for the above is that the "customers,orders" filter will be applied to the groupId ds_orders1 and the lone orders will be applied to ds_orders2. **sourceResource**: The derivedFilterPath may now be used with the explicit variable sourceResource as long as sourceResource points to a CONTAINER/FOLDER resource. If sourceResource points to a TABLE/VIEW resource then derivedFilterPath is ignored. For example: • sourceResource=/shared/lab00/Physical/Metadata/ ds_orders1 • derivedFilterPath=customers,orders The result for the above is that customers and orders are the only views generated.	
IN	**typeDefProcName –** The name of the type definition procedure. e.g. TypeDefinitions. If null, then the default name	VARCHAR

Direction	Parameter Name	Parameter Type
	'TypeDefinitions' is used.	
OPTION 1	**Explicit Folders:** Use explicit sourceResource and publishToFolder parameters with derivedFilterPath being optional.	
IN	**sourceResource –** The source folder in CIS to begin searching for views to publish to a Composite data source or to another folder or the specific source View or Procedure to publish to a Composite data source or to another folder. If this is set it supercedes layerType, inGroupIDs and derivedFilterPath	
IN	**generateToFolder –**This is the full path to the folder in which to generate the views. This is only required if option 1: targetResource is provided. If sourceResource is not blank, then it is used and groupIds and derivedFilterPath are ignored	
OPTION 2	**ConfigureStartingFolders:** If sourceResource is blank, then groupIds must be set with derivedFilterPath being optional.	
IN	**layerType** • PV=Physical Views – physical views • FV=Formatting Views - formatting views • BV=Business Views - business views (only single source) • LV=Logical Views - logical views (only single source) • CV=Client Views - client views (only single source) • CP=Client Published - client published views (only single source) • DB=Database link – generate a Composite database link	VARCHAR
IN	**groupIds** • This is a comma separate list group ids to process from the ConfigureStartingFolders. • Pass in null to select all groupIds.	LONGVARCHAR
OUT	**result** CURSOR TypeDefinitions.generateCRUDRow (datasourceName VARCHAR(255), projectFolderName VARCHAR(255), greatGrandParentName VARCHAR(255), grandParentName VARCHAR(255), parentName VARCHAR(255), containerName VARCHAR(255), containerType VARCHAR(255), name VARCHAR(255),	CURSOR

Direction	Parameter Name		Parameter Type
	logicalName	VARCHAR(255),	
	logicalType	VARCHAR(255),	
	logicalStatus	VARCHAR(255),	
	containerPath	TypeDefinitions.pathType	
	duration	INTERVAL HOUR TO SECOND	
	);		

Rebind Scripts: Rebind All Resources

These scripts are used for rebinding resources from one folder to another.

29. **rebindAllResources** – This procedure is used to rebind all of the resources (Views) in a given starting source folder to a target rebind folder. For example, if all of the views in the Formatting Views layer are pointing to a particular data source and you want to rebind them to point to a different data source folder then this procedure will accomplish that task. This may be useful when redeploying from Dev to Test to Production or simply rebinding to a different development instance of the database. Rules:

    1) If a resource in the folder has both the source and the target sources present, it will use rebindResource to do an explicit rebind.

    2) If a resource in the folder does not have the source present, it will rebind using explicit text modification techniques instead of rebindResource. The following text modification techniques are supported for the given resource type:

        resourceType = 'TABLE'

        subtype = 'SQL_TABLE' -- Regular View not a database table

        resourceType = 'PROCEDURE'

        subtype = 'SQL_SCRIPT_PROCEDURE' -- Custom Procedure or Parameterized query

        subtype = 'EXTERNAL_SQL_PROCEDURE' -- Packaged Query Procedure

        subtype = 'BASIC_TRANSFORM_PROCEDURE' -- XSLT Basic Transformation definition

        subtype = 'XSLT_TRANSFORM_PROCEDURE' -- XSLT Transformation text

        subtype = 'STREAM_TRANSFORM_PROCEDURE' -- XSLT Stream Transformation text

3) If a resource in the folder does not have the target present, that is an error and an exception is raised.

Direction	Parameter Name	Parameter Type
IN	startingResourceFolder	/shared/ASAssets/Utilities/TypeDefinitions.pathType
IN	rebindFromFolder	/shared/ASAssets/Utilities/TypeDefinitions.pathType
IN	rebindToFolder	/shared/ASAssets/Utilities/TypeDefinitions.pathType
OUT	success – 0 or 1	BIT
OUT	faultResponse- null if succesful otherwise contains a fault resposne.	LONGVARCHAR