How To Use Data Abstraction Best Practices Dynamic File Framework

An Open Source Asset for use with TIBCO® Data Virtualization

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

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
2018Q1	03/20/2018	Mike Tinius	Created new
2019Q1	01/25/2019	Mike Tinius	Release 2019Q1 - no changes.
2019Q200	06/13/2019	Mike Tinius	Release 2019Q200 – no changes.
2019.300	08/01/2019	Mike Tinius	Release 2019Q300 – no changes.
2020.200	03/12/2020	Mike Tinius	Release 2020Q200 – no changes.
2020.201	05/12/2020	Mike Tinius	Bug fix spaces in path. Query optimization example. Auto-delete files.
2020.300	08/20/2020	Mike Tinius	Bug fix for SQL Server blocking issue
2020.400	12/12/2020	Mike Tinius	Updated "Learn" documentation. Fixes in View Generation and Privilege Scripts modules.

Related Documents

Name	Version
How To Use Utilities.pdf	2020Q402
How To Use Data Abstraction Best Practices View Generation.pdf	2020Q400
How To Test Data Abstraction Best Practices View Generation.pdf	2020Q400
How To Learn Data Abstraction Best Practices View Generation.pdf	2020Q400
How To Use Data Abstraction Best Practices Manage Annotations.pdf	2020Q200
How To Use Data Abstraction Best Practices Privilege Scripts.pdf	2020Q400
How To Use Data Abstraction Best Practices Dynamic File Framework.pdf	2020Q300

Supported Versions

Name	Version
TIBCO® Data Virtualization	7.0 or later
AS Assets Utilities open source	2020Q402 or later

Table of Contents

1 Introduction

Purpose

The purpose of Best Practices Dynamic File Framework Scripts is to provide a framework for a user or application to upload files to the Data Virtualization server and publish them as views so they can be queried and/or joined with other DV resources.

The Dynamic File Framework Scripts provide a mechanism to process the files and publish the files as views. However, it is up to the developers of this framework to provide a means to upload the actual files to the DV server. Once the files are on the server, this framework will provide the necessary capabilities to list, add, remove, cleanup, auto-publish and rebuild. The Dynamic File Framework supports the following file types: Excel (.xls and .xlsx) and CSV (.txt and .csv). Because file names may overlap, it is required that the filename be prefixed with the username and dash such that the file name format is username-filename.extension. The username portion of the file must be unique and is correlated with an email.

The diagram below shows the Data Abstraction Best Practices layers. The developer of this framework may choose which layers to auto-create the views. It is permitted to introspect the physical data source and publish straight to the published database layer or create a view in each layer including Formatting, Business/Logical, Application/Views and finally the published database layer.

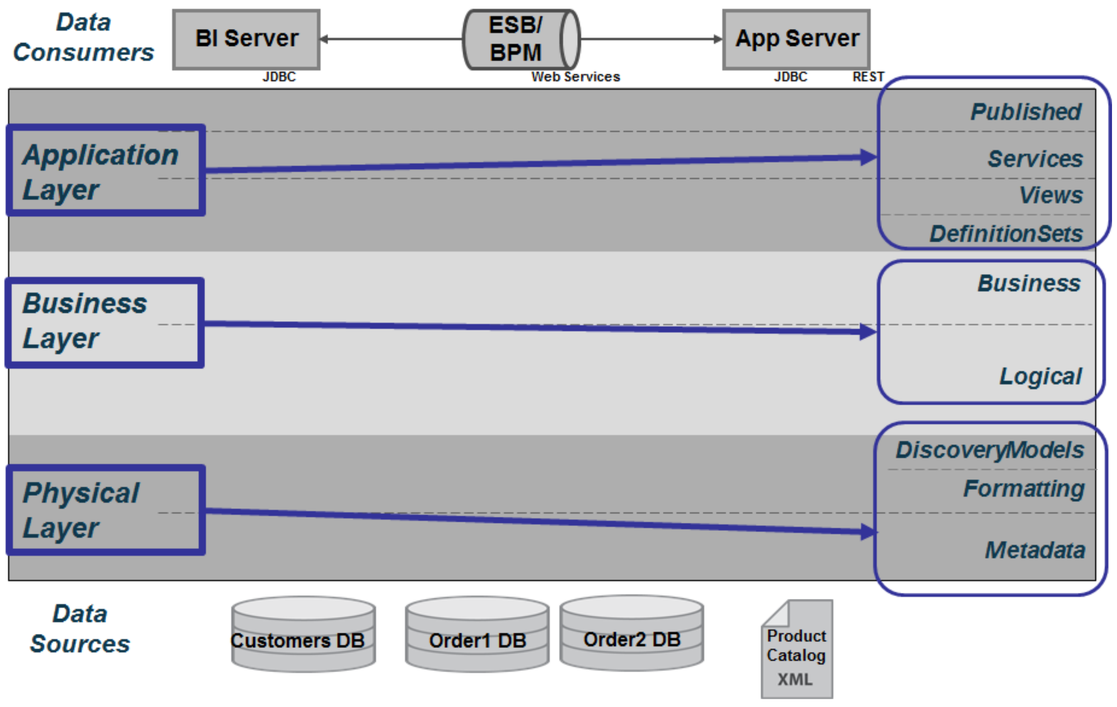


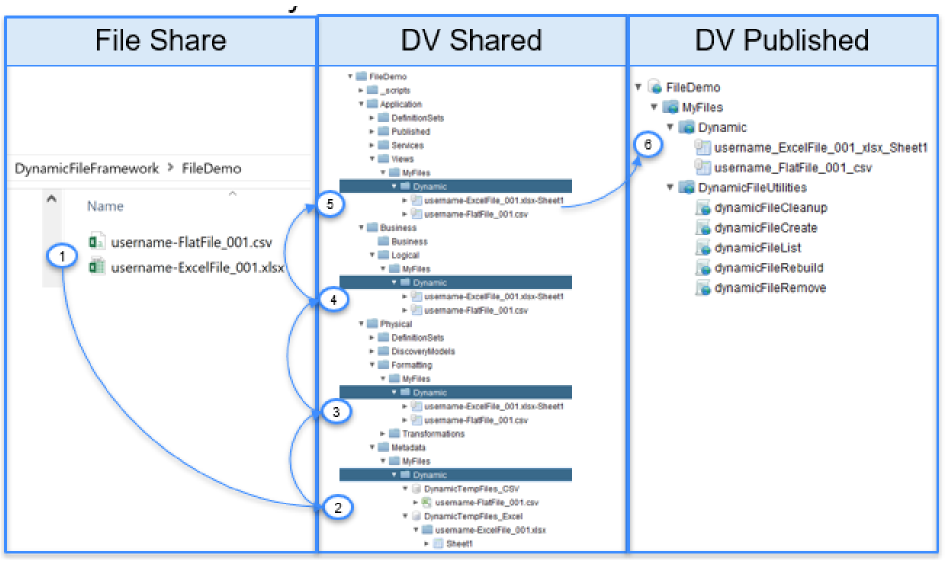
Figure one: Technical Data Abstraction Layers

Features of Dynamic File Framework

This section describes the features of the dynamic file framework.

1. Auto-publish files from file system to DV published database folder
2. Auto-delete files by removing them from the file system
3. Invoke API via JDBC or Web Services
    a. dynamicFileList
    b. dynamicFileCreate
    c. dynamicFileRemove
    d. dynamicFileCleanup
    e. dynamicFileRebuild
    f. dynamicFileAutoPublish
4. Automatically sends email when one of the API actions are invoked.
5. Ability to configure file framework at the project level in order to apply permissions on folders.
6. Logs activity in a database table.

Diagram of how the file to published view works:



1. Landing Zone described at the subject area level.
2. Data Source (pre-created) which imports all objects from #1.
3. Formatting Folder that contains all objects from the file Share Folder.
4. Business Layer folder that contains a view of all objects in the Dynamic Formatting folder.
5. Application Layer Views folder.
6. Published area schema. Views are ODATA compliant [no special characters]

Audience

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

- Data Virtualization Administrators – provides a guide for installation.
- Architects – provides the data abstraction architecture.
- Data professionals – provides background on the published views and usage.
- Operations users – provides insight into triggers and procedures that are executed.
- Project Managers – provides general information on dynamic file framework best practices.

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)

Pre-Requisites

Follow the steps below to create a new project.

1. LDAP groups have been brought into the target environment. /shared/ASAssets/Utilities have been installed and configured for 2020Q200. There are new utilities in this release required by the Dynamic File Framework.

2 Configuration

How to Configure

This section provides information on how to configure the Dynamic File Framework Scripts.

Dynamic File Framework Script Configuration Summary

1. Configure DV email
2. Configure data sources and constants
3. Create the database tables and sequences
4. Add emails for each user using the framework
5. Turn on trigger
6. Configure the customized framework for each project

Dynamic File Framework Installation

1. DV EMAIL:

   1.1. Configure the Data Virtualization Email.

   1.2. In Studio, go to Administration → Server → Configuration → E-Mail

2. DV LOOPBACK DATA SOURCE:

   2.1. Modify the "admin" password for the data source DYNAMIC_FILE_LOCAL_LOOPBACK

   2.2. The loopback is required so that execution of "triggered" procedure run as a user with proper rights to execute internal procedures for creating and dropping views and assigning privileges.

3. DATABASE + FILE SYSTEM

   3.1. Provide a data source to be used for the Dynamic File Framework queue and email tables.

   3.2. You may choose to use the out-of-the-box Postgres data source with the built in “ciscache” schema. For this, you will have to set the connection information for this data source:
   /shared/ASAssets/BestPractices_v81/DynamicFileFramework/DYNAMIC_FILE_QUEUE_DS

   3.3. You may also choose to use your own database but it must be Oracle, SQL Server or Postgres. A data source must be provided within DV. The path, catalog, schema and tablespace/filegroup are configured in the “constants” procedure in step 4.

3.4. For DV cluster environments it is “NOT” recommended to use the built-in postgres database. Supply a data source that is accessible to all nodes in the cluster. All nodes require the same access to both database and file system. Therefore, the file system needs to be a shared file system between all nodes of the cluster.

4. CONSTANTS:

4.1. Modify the constants file and set the following accordingly

SET ENVIRONMENT_NAME = 'DEV'; -- [mandatory] The current environment nickname. Set a unique environment name suchas as [DEV, TEST, QA, PROD].

SET EXECUTE_DDL_PACKAGE_PATH ='/shared/ASAssets/BestPractices_v81/DynamicFileFramework/00_ExecuteDDL'; -- [mandatory] The location of the Execute DDL package path

SET EXECUTE_DML_PACKAGE_PATH = bestPracticesRootPath||'/DynamicFileFramework/02_display_DYNAMIC_FILE_QUEUE_pq'; -- [mandatory] The location of the 02_display_DYNAMIC_FILE_QUEUE_pq package path

SET DATASOURCE_PATH = '/shared/ASAssets/BestPractices_v81/DynamicFileFramework/DYNAMIC_FILE_QUEUE_DS'; -- [mandatory] CIS datasource path. Supported: Oracle, SQL Server and Postgres

SET CATALOG_NAME = null; -- [optional] Catalog name for SQL Server

SET SCHEMA_NAME = 'ciscache'; -- [mandatory] Schema name

SET TABLESPACE_FILEGROUP = null; -- [optional] Tablespace name if Oracle or Postgres and file group name if SQL Server.

SET PROCESS_WAIT_TIME_SECONDS = '5'; -- [mandatory] The number of seconds to wait in between processing requests.

SET DEFAULT_EMAIL = 'admin-email@company.com'; -- [mandatory] Default email to send exceptions to.

SET DEFAULT_RETENTION_POLICY_DAYS = '30'; -- [mandatory] Default retention policy. Delete files and CIS resources older than x days. This is a numeric value that will be converted from text to integer.

SET DYNAMIC_FILE_SEQ = 'DYNAMIC_FILE_SEQ'; -- [DO NOT CHANGE] Dynamic File Sequence Name.

SET DYNAMIC_FILE_EMAIL = 'DYNAMIC_FILE_EMAIL'; -- [DO NOT CHANGE] Dynamic File Email Table Name.

SET DYNAMIC_FILE_QUEUE = 'DYNAMIC_FILE_QUEUE'; -- [DO NOT CHANGE] Dynamic File Queue Table Name.

SET CSV_SUPPORTED_FILE_EXTENSIONS = '.csv,.txt'; -- [DO NOT CHANGE] CSV Supported File Extensions

SET EXCEL_SUPPORTED_FILE_EXTENSIONS = '.xls,.xlsx'; -- [DO NOT CHANGE] Excel Supported File Extensions

5. EXECUTE DDL

5.1. Create the sequence generator and queue table and rebind 00_ExecuteDDL

5.2. Execute ***01_pqCreateDrop_dynamic_file_tables*** and provide the following:

This procedure uses information from the "constants" procedure to dynamically construct the DDL. Using the parameters below, you can achieve different combinations of dropping and creating or nothing at all.

Input:

executeDDL:	y=execute DDL,	n=print DDL
dropResources:	y=Drop resources,	n=don't drop resources
createResources:	y=Create resources,	n=don't create resources

What gets created/dropped:

Sequence Name:	DYNAMIC_FILE_SEQ
Table Name:	DYNAMIC_FILE_QUEUE
Table Name:	DYNAMIC_FILE_EMAIL

5.3. The script rebinds the 00_ExecuteDDL to point to the data source.

5.4. The script introspects resources on the target data source.

Add the table DYNAMIC_FILE_QUEUE

Add the table DYNAMIC_FILE_EMAIL

6. Rebind 02_display_DYNAMIC_FILE_QUEUE_pq

6.1. The script rebinds the 00_ExecuteDDL to point to the datasource.

7. ADD USER EMAILS:

7.1. Add emails for each user that will be invoking the procedures

7.2. Invoke /shared/ASAssets/BestPractices_v81/DynamicFileFramework/Helper/emailAddUser

7.3. Add a general group email for the user "admin" so that. if files are generically created/deleted at least the admin will be notified.

8. CENTRAL PROCESSING TRIGGER:

8.1. Turn on the Trigger: DynamicFileFrameworkProcessTrigger

8.2. Set the periodic refresh rate according to your policy. Default is 15 minutes. This is how often the trigger will wake up and look for rows in DYNAMIC_FILE_QUEUE table to process. If it finds no rows then no work is done. If it does find rows, the requests are executed in a single-threaded fashion with the default time delay applied in between requests as set in constants.PROCESS_WAIT_TIME_SECONDS. The purpose of the time-delay is so that DV does not get overwhelmed by processing files.

9. CONFIGURE PROJECT:

9.1. Proceed to either OPTION 1 (automated install) or OPTION 2 (manual install) below to generate the Dynamic File Framework within a project folder for a user application.

9.2. Consider this as the project interface procedures for the Dynamic File Framework implementation core code. The interface procedures provide a way in which a specific project can offer a simplified interface to its users. Since each project has a generated "constants" procedure with the project values stored, it can keep the actual interface code to a minimum and pass the "constants" values into the core Dynamic File Framework.

OPTION 1 - Automated template creation (recommended)

1. Locate the procedure:
   /shared/ASAssets/BestPractices_v81/DynamicFileFramework/Example/example_Create_or_Remove

   1.1. Make a copy of it and paste it into your project folder location of your choice.

   1.2. Rename it to give it a meaningful name associated with the target project.

   1.3. Modify the paths and other variables for your use case.

   1.4. Execute it with a 1 to create the Dynamic File Framework resources in your project path.

   1.5. Execute it with a 0 to destroy the Dynamic File Framework resources in your project path. [Validating the removal works.]

   1.6. Execute it with a 1 to create the Dynamic File Framework resources in your project path.

2. Proceed to the "***Executing the Dynamic File Framework***" section

OPTION 2 - Manual template creation (not recommended)

1. Copy the Dynamic folder from ***DynamicFileFramework/Templates/Application/Services/Dynamic*** and paste it into your project's /Application/Services folder.

   ex. /shared/base_project_path/business_area/subject_area/Application/Services

2. Modify the variables in your new location "constants" procedure

   NOTE: Take note of mandatory and optional variable settings within the constants.

   -- Standard application default settings [ORG, PRJ1, S1, ORGDB]

   ```
   SET ORGANIZATION = '';      -- [mandatory] Identifies your organization for tracking/filtering purposes.
   SET PROJECT_NAME = '';      -- [mandatory] This is your project folder name and is used for tracking/filtering purposes.
   SET SUBPROJECT_NAME    = ''; -- [optional] Leave blank if your project folder does not have a sub project
   SET PUBLISHED_DATABASE = ''; -- [mandatory] This is the published database name.
   -- Published Schema Path: e.g. /services/databases/published_data_source/catalog/Dynamic : Dynamic is recommended for the schema name
   ```

SET PUBLISHED_SCHEMA_PATH = ''; -- [mandatory] The full path to the published Dynamic schema within your published databases

-- Physical Metadata Layer Path

SET PHYSICAL_METADATA_LAYER = '';   -- [mandatory] The full path to the physical metadata Dynamic folder where the dynamic file data sources reside. Set to the default null for no view generation in the layers. Otherwise set to Option 2 to generate views in the layers.

SET APPLICATION_LAYER = null;   -- Option 1: null, Option 2: Full path to /Physical/Formatting/Dynamic folder

SET   BUSINESS_LAYER = null;   -- Option 1: null, Option 2: Full path to /Business/Logical/Dynamic folder

SET FORMATTING_LAYER = null;   -- Option 1: null, Option 2: Full path to /Application/Views/Dynamic folder

-- Allows the invoking interface procedure to control the behavior of the implementation procedure.

SET ALLOW_NULL_FILE_NAME = '0';   -- 0=Do not allow user to pass in null/empty for the file name. Throw an exception.

-- 1=Allow the user to pass in null/empty for the file name which results in picking up all unclaimed (not introspected) files in the file system for the invoking user.

-- Default retention policy.

SET RETENTION_POLICY_DAYS = '30';   -- [mandatory] Delete files and CIS resources older than x days. This is a numeric value that will be converted from text to integer.

2.1. Save the procedure

3. Copy the Dynamic folder from Templates/Physical/Metadata/Dynamic and paste it into your project's /Metadata folder.

   Note: In both cases, if file system security is being used on the DV server, then you will need to put in the "File System Security Root Mapping Name" associated with the file system location. Work with the DV administrator to set up Root Mapping Names and paths.

```
E.g.  /shared/BusArea/SubjectArea/Physical/Metadata
                                          /Dynamic
                                                  /DynamicTempFiles_CSV
                                                  /DynamicTempFiles_Excel
```

3.1. Open the data source "Dynamic/DynamicTempFiles_CSV"

3.1.1. Edit the Root Path to point to the DV file system path where your project files are stored or use the root mapping name if applicable.

3.1.2. Test the connection and execute the Add/Remove Tables to make sure the files are visible.

3.2. Open the data source "Dynamic/DynamicTempFiles_Excel"

3.2.1. Edit the Root Path to point to the DV file system path where your project files are stored or use the root mapping name if applicable.

3.2.2. Test the connection and execute the Add/Remove Tables to make sure the files are visible.

4. Create a published relational catalog and schema
    4.1. Create a published database if it does not exist as per constants.PUBLISHED_DATABASE.
    4.2. Create a catalog name according to your project and subproject
    4.3. Create a schema name called Dynamic.
    4.4. Publish the procedures to the published /services/databases/published_datasource/catalog/DynamicFileUtilities schema
    4.5. Select the following procedures. You will need to Control-Click to select them all
        4.5.1. dynamicFileCleanup
        4.5.2. dynamicFileCreate
        4.5.3. dynamicFileList
        4.5.4. dynamicFileRebuild
        4.5.5. dynamicFileRemove
    4.6. Right-click on the selected procedures and choose Publish
    4.7. Browse and select the /services/databases/published_datasource/catalog/DynamicFileUtilities schema and click OK

5. Create a web service
    5.1. Create a folder that mimics the catalog and schema name from the relational datasource.

        e.g. /services/webservices/published_datasource/catalog/DynamicFileUtilities

    5.2. Create a web service in /services/webservices/published_datasource/catalog/DynamicFileUtilities called Utilities
    5.3. Publish the procedures to the published /services/webservices/published_datasource/catalog/DynamicFileUtilities/Utilities web service
        5.3.1. Select the following procedures. You will need to Control-Click to select them all
            5.3.1.1. dynamicFileCleanup
            5.3.1.2. dynamicFileCreate
            5.3.1.3. dynamicFileList
            5.3.1.4. dynamicFileRebuild
            5.3.1.5. dynamicFileRemove
        5.3.2. Right-click on the selected procedures and choose Publish

5.3.3. Browse and select the Utilities web service and click OK

6. Test the procedures for create, remove and list
   6.1. SFTP files to your designated file system location on the target DV server host.
   6.2. Create files/views
      6.2.1. Execute the published "dynamicFileCreate" procedure from a 3rd party tool such as DB Visualizer. File Views should be visible now to execute queries from in the DV published schema
   6.3. List files/views
      6.3.1. Execute the published "dynamicFileList" procedure from a 3rd party tool such as DB Visualizer. A listing of files and published views should be displayed.
   6.4. Remove the file from the target DV server file system.
      6.4.1. Execute the published "dynamicFileRemove" procedure from a 3rd party tool such as DB Visualizer. Files and Views should now be removed and therefore the same query should fail that was previously executed.

7. Turn on both triggers if applicable:
   7.1. DynamicFileFrameworkAutoPublishTrigger - Will automatically detect files in the file system and publish them if turned on.
   7.2. DynamicFileFrameworkCleanupTrigger - Will automatically detect files that are out of policy and remove the views and files.
8. Proceed to the “***Executing the Dynamic File Framework***” section

3 Executing the Dynamic File Framework

Introduction

This section provides guidance on how to invoke from a 3rd party tool such as Toad or DB Visualizer. Once you have completed OPTION 1 or OPTION 2 for the installation/setup, now you are ready to start using the Dynamic File Framework.

Instructions

1. The basic flow looks like this

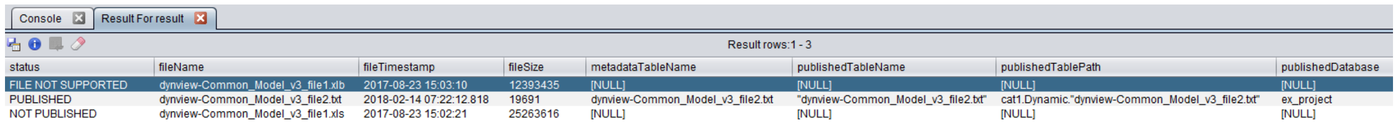
   1.1. Create either an ODBC or JDBC connection from your favorite 3rd party tool to the target DV instance.
   1.2. Open your favorite Query tool
   1.3. Execute one or more of the following using the sample project as a guideline:

LIST FILES - dynamicFileList():

The purpose of the "dynamic file list" is to show what files are published as views, not published at all and which ones are not supported.

DATABASE: ex_project - ODBC/JDBC connection is made

SELECT * FROM "cat1"."Dynamic"."dynamicFileList"()



status	fileName	fileTimestamp	fileSize	metadataTableName	publishedTableName	publishedTablePath	publishedDatabase
FILE NOT SUPPORTED	dynview-Common_Model_v3_file1.xlb	2017-08-23 15:03:10	12393435	[NULL]	[NULL]	[NULL]	[NULL]
PUBLISHED	dynview-Common_Model_v3_file2.txt	2018-02-14 07:22:12.818	19691	dynview-Common_Model_v3_file2.txt	"dynview-Common_Model_v3_file2.txt"	cat1.Dynamic."dynview-Common_Model_v3_file2.txt"	ex_project
NOT PUBLISHED	dynview-Common_Model_v3_file1.xls	2017-08-23 15:02:21	25263616	[NULL]	[NULL]	[NULL]	[NULL]

Output Messages:

Status Message	Rule
FILE NOT SUPPORTED	File type is not supported. It will be marked for removal.
NOT PUBLISHED | METADATA=no | FILE=yes	File was just added to file system but the process has not picked it up to process it yet.
PUBLISHED | METADATA=yes | FILE=yes	File is published. It exists in the data source metadata. It exists in the file system.
NOT PUBLISHED | METADATA=yes | FILE=yes	File is not published. It exists in the data source metadata. File exists in the file system.
PUBLISHED | METADATA=yes | FILE=no	File is published. It exists in the data source metadata. It does not exist in the file system. Clean-up will mark it for removal.
NOT PUBLISHED | METADATA=yes | FILE=no	File is not published. It exists in the data source metadata. It does not exist in the file system. Clean-up will mark it for removal.

ADD FILE - dynamicFileCreate():

The purpose of “dynamic file create” is to add/create the views for the specified file residing in the file system. This procedure queues up the request into the DYNAMIC_FILE_QUEUE table. The request is processed by the framework procedure “dynamicFileQueueProcess” which is triggered by the framework trigger “DynamicFileFrameworkProcessTrigger”.

DATABASE: ex_project - ODBC/JDBC connection is made

How to register a file to be created as the actual user who owns the file:

This scenario is useful for users who are directly connecting to DV to upload files to their file system area and invoking the DV Dynamic File Framework API to register the file creation.

SELECT * FROM "cat1"."Dynamic"."dynamicFileCreate"('user1-Common_Model_v3_file2.xlsx')

How to register a file to be created as a proxy user (service account) who does not own the file:

This scenario comes into play when there is a service account such as an application that is controlling the interface between the user and the DV layer. For example, there may be a front-end web application that the users log into to upload their files. Therefore, the service account of the application is the proxy user which both uploads the file and invokes the DV Dynamic File Framework API's to register the file creation.

SELECT * FROM "cat1"."Dynamic"."dynamicFileCreate"('USERNAME[user1]user1-Common_Model_v3_file2.xlsx')

Queuing Messages:

'FILE ADD QUEUED'	actiontype='A'
'INVALID FILENAME'	actiontype='A','R'
'DUPLICATE REQUEST IGNORED'	actiontype='A','R','C'

Email Messages:

'UNKNOWN'

'FILE ADDED. NO VIEWS PUBLISHED'

'FILE ADDED. VIEWS PUBLISHED'

'FILE ADDED. VIEWS ALREADY EXISTS'

'FILE ADDED. VIEWS OVERWRITTEN'

'FILE DOES NOT EXIST'

- The following occurs when the file being added contains no Excel sheets and/or no columns and no data:

'FILE REMOVED. NO COLUMNS DETECTED. VIEWS NOT PUBLISHED'

'UNABLE TO REMOVE FILE. NO COLUMNS DETECTED. VIEWS NOT PUBLISHED'

'FILE DOES NOT EXIST. NO COLUMNS DETECTED. VIEWS NOT PUBLISHED'

'ERROR: Data source type not supported: '+<dataSourceType>

'ERROR: File Name is required.' - occurs when (allowNullFileName = 0 and (fileName IS NULL OR LENGTH(fileName) = 0))

REMOVE FILE - dynamicFileRemove():

The purpose of the "dynamic file remove" is to remove the file and views for the specified file residing in the file system. This procedure queues up the request into the DYNAMIC_FILE_QUEUE table. The request is processed by the framework procedure "dynamicFileQueueProcess" which is triggered by the framework trigger "DynamicFileFrameworkProcessTrigger".

DATABASE: ex_project - ODBC/JDBC connection is made

How to register a file to be removed as the actual user who owns the file:

SELECT * FROM "cat1"."Dynamic"."dynamicFileRemove"('user1-Common_Model_v3_file2.xlsx')

How to register a file to be removed as a proxy user (service account) who does not own the file:

SELECT * FROM "cat1"."Dynamic"."dynamicFileRemove"('USERNAME[user1]user1-Common_Model_v3_file2.xlsx')

Queuing Messages:

'FILE REMOVE QUEUED'	actiontype='R'
'INVALID FILENAME'	actiontype='A','R'
'DUPLICATE REQUEST IGNORED'	actiontype='A','R','C'

Email Messages:

'FILE REMOVED. ' + 1 of the following:

'PUBLISHED VIEWS REMOVED'

'VIEWS NOT PUBLISHED'

'UNABLE TO REMOVE FILE. ' + 1 of the following:

'PUBLISHED VIEWS REMOVED'

'VIEWS NOT PUBLISHED'

'FILE DOES NOT EXIST. ' + 1 of the following:

'PUBLISHED VIEWS REMOVED'

'VIEWS NOT PUBLISHED'

'FILE NOT ASSOCIATED WITH USERNAME=<username>'

'FILE DOES NOT EXIST. FILE NOT ASSOCIATED WITH USERNAME=<username>'

CLEANUP FILES - dynamicFileCleanup():

The purpose of the “dynamic file cleanup” is to clean-up old files that have a file timestamp beyond the retention policy. The file and the associate views are removed. This procedure queues up the request into the DYNAMIC_FILE_QUEUE table. The file cleanup procedure may be invoked manually or it may be triggered once a day by the trigger “DynamicFileFrameworkCleanupTrigger”. The request is processed by the framework procedure “dynamicFileQueueProcess” which is triggered by the framework trigger “DynamicFileFrameworkProcessTrigger”.

DATABASE: ex_project - ODBC/JDBC connection is made

How to register a cleanup for either an actual user or proxy service account user:

SELECT * FROM "cat1"."Dynamic"."dynamicFileCleanup"()

Queuing Messages:

'FILE CLEAN-UP QUEUED'	actiontype='C'
'DUPLICATE REQUEST IGNORED'	actiontype='A','R','C'

Email Messages:

'RETENTION POLICY EXCEEDED. ' + dynamicFileRemove message

or

'FILE NOT SUPPORTED. ' + dynamicFileRemove message

- dynamicFileRemove messages:

'FILE REMOVED. ' + 1 of the following:

'PUBLISHED VIEWS REMOVED'

'VIEWS NOT PUBLISHED'

'UNABLE TO REMOVE FILE. ' + 1 of the following:

'PUBLISHED VIEWS REMOVED'

'VIEWS NOT PUBLISHED'

'FILE DOES NOT EXIST. ' + 1 of the following:

'PUBLISHED VIEWS REMOVED'

'VIEWS NOT PUBLISHED'

'FILE NOT ASSOCIATED WITH USERNAME=<username>'

'FILE DOES NOT EXIST. FILE NOT ASSOCIATED WITH USERNAME=<username>'

REBUILD FILES - dynamicFileRebuild():

The purpose of “dynamic file rebuild” is to add/create the views for all unpublished files residing in the file system. This procedure queues up requests for each into the DYNAMIC_FILE_QUEUE table. The request is processed by the framework procedure “dynamicFileQueueProcess” which is triggered by the framework trigger “DynamicFileFrameworkProcessTrigger”.

DATABASE: ex_project - ODBC/JDBC connection is made

How to register a rebuild for either an actual user or proxy service account user:

SELECT * FROM "cat1"."Dynamic"."dynamicFileRebuild"()

Queuing Messages:

'FILE ADD QUEUED'	actiontype='A'
'INVALID FILENAME'	actiontype='A','R'
'DUPLICATE REQUEST IGNORED'	actiontype='A','R','C'

Email Messages:

'UNKNOWN'

'FILE ADDED. NO VIEWS PUBLISHED'

'FILE ADDED. VIEWS PUBLISHED'

'FILE ADDED. VIEWS ALREADY EXISTS'

'FILE ADDED. VIEWS OVERWRITTEN'

'FILE DOES NOT EXIST'

- The following occurs when the file being added contains no Excel sheets and/or no columns and no data:

'FILE REMOVED. NO COLUMNS DETECTED. VIEWS NOT PUBLISHED'

'UNABLE TO REMOVE FILE. NO COLUMNS DETECTED. VIEWS NOT PUBLISHED'

'FILE DOES NOT EXIST. NO COLUMNS DETECTED. VIEWS NOT PUBLISHED'

'ERROR: Data source type not supported: '+<dataSourceType>

'ERROR: File Name is required.' - occurs when (allowNullFileName = 0 and (fileName IS NULL OR LENGTH(fileName) = 0))

AUTOMATED PUBLISH (ADD) - dynamicFileAutoPublish():

The purpose of “dynamic file auto-publish” is to automatically add/create the views for any new files residing in the file system. This procedure queues up the request for each new file found into the DYNAMIC_FILE_QUEUE table. The request is processed by the framework procedure “dynamicFileQueueProcess” which is triggered by the framework trigger “DynamicFileFrameworkProcessTrigger”.

Turn on the trigger "DynamicFileFrameworkAutoPublishTrigger".

Check the schedule to determine how often you want it to wake up and look for files to add to the queue. The default is every 5 minutes at nn:04. The reason for 04 is that the "DynamicFileFrameworkProcessTrigger" wakes up on the :00 or :05 or :15 so that the processing of the Publish trigger would be picked soon after it was registered.

Queuing Messages:

'FILE ADD QUEUED'	actiontype='A'
'INVALID FILENAME'	actiontype='A','R'
'DUPLICATE REQUEST IGNORED'	actiontype='A','R','C'

Email Messages:

'UNKNOWN'

'FILE ADDED. NO VIEWS PUBLISHED'

'FILE ADDED. VIEWS PUBLISHED'

'FILE ADDED. VIEWS ALREADY EXISTS'

'FILE ADDED. VIEWS OVERWRITTEN'

'FILE DOES NOT EXIST'

- Occurs when the file being added contains no Excel sheets and/or no columns and no data

'FILE REMOVED. NO COLUMNS DETECTED. VIEWS NOT PUBLISHED'

'UNABLE TO REMOVE FILE. NO COLUMNS DETECTED. VIEWS NOT PUBLISHED'

'FILE DOES NOT EXIST. NO COLUMNS DETECTED. VIEWS NOT PUBLISHED'

'ERROR: Data source type not supported: '+<dataSourceType>

'ERROR: File Name is required.' - occurs when (allowNullFileName = 0 and (fileName IS NULL OR LENGTH(fileName) = 0))

AUTOMATED CLEANUP - dynamicFileCleanup():

The purpose of the "dynamic file cleanup" is to automatically clean-up old files that have a file timestamp beyond the retention policy. The file and the associate views are removed. This procedure queues up the request into the DYNAMIC_FILE_QUEUE table. The request is processed by the framework procedure "dynamicFileQueueProcess" which is triggered by the framework trigger "DynamicFileFrameworkProcessTrigger".

Turn on the trigger "DynamicFileFrameworkCleanupTrigger".

Check the schedule to determine how often you want it to wake up and schedule a cleanup into the queue. The default is once a day at 4:59 am. This is on an odd number for the same reason as the publish trigger is so the processing would be picked up soon after it was registered.

Queuing Messages:

'FILE CLEAN-UP QUEUED'	actiontype='C'
'DUPLICATE REQUEST IGNORED'	actiontype='A','R','C'

Email Messages:

'RETENTION POLICY EXCEEDED. ' + dynamicFileRemove message

or

'FILE NOT SUPPORTED. ' + dynamicFileRemove message

- dynamicFileRemove messages:

'FILE REMOVED. ' + 1 of the following:

    'PUBLISHED VIEWS REMOVED'

    'VIEWS NOT PUBLISHED'

'UNABLE TO REMOVE FILE. ' + 1 of the following:

    'PUBLISHED VIEWS REMOVED'

    'VIEWS NOT PUBLISHED'

'FILE DOES NOT EXIST. ' + 1 of the following:

    'PUBLISHED VIEWS REMOVED'

    'VIEWS NOT PUBLISHED'

'FILE NOT ASSOCIATED WITH USERNAME=<username>'

'FILE DOES NOT EXIST. FILE NOT ASSOCIATED WITH USERNAME=<username>'

4 Optimizing Queries Using Dynamic File Framework

Introduction

This section describes the best practice for optimizing queries when using files. Generally speaking, the data being brought from files is typically used for lookup or augmentation of data that exists in tables. It is very fluid and changes rapidly. File data is often used because it may be temporary in nature.

Optimization

The following technique is used when the relational table is much larger than the file data. The example query would be executed by a client tool and passed into data virtualization through JDBC or ODBC. There are a few things to point out about the query below:

1. The file always appears first in the query.
2. The data virtualization optimizer hint goes between the INNER and the JOIN keywords.
3. It's always a good idea to have a where clause selecting the file id column where not null because files often have rows of empty or null data.

Specific hint:

```
{OPTION SEMIJOIN="True", LEFT_CARDINALITY="3000", RIGHT_CARDINALITY="100000"}
```

SQL Statement:

```
Select <column_list>

From <catalog>.<schema>."username-filename.csv" F

Inner { OPTION SEMIJOIN="True", LEFT_CARDINALITY="3000",
RIGHT_CARDINALITY="100000" } Join

<catalog>.<schema>.<relational_table> T

On F.idcolumn = T.id

Where F.id is not null
```

What to expect with different queries:

Query 1: File first. Uses optimizer hints. Fast response. Low memory usage.

Query 2: File first. No optimizer hints. Moderate response. Low memory usage.

Query 3: File last. No optimizer hints. Slow response. High memory usage.