Advanced Services Assets

Cache Management Framework

Data Virtualization Business Unit Advanced Services

October 2014

TABLE OF CONTENTS

DOCUMENT CONTROL

Version History

Version	Date	Author	Description
0.1	09/07/2010	Calvin Goodrich	Initial revision
0.2	03/06/2013	Calvin Goodrich	Updated to take advantage of the incremental caching framework of CIS 6.2.1
0.3	05/13/2014	Calvin Goodrich	Updated to reflect collection of AS tools and utilities into /shared/ASAssets
1.0	11/04/2014	Mike Tinius	Created the Cache Framework

Related Documents

Document	Date	Author
6.2 SP1 Users Guide (or later)		

Data Virtualization Business Unit (DVBU) Products Referenced

DVBU Product Name	Version
Composite Information Server 6.2	6.2.1 or later
AS DVBU Utilities	2014Q4307

INTRODUCTION

Purpose

With the release of CIS 6.2 SP1, CIS now features a framework to support the development of custom full and pull-based incremental caching solutions. This document describes such a solution developed in the field by the AS team. At this time it only supports full and incremental caching of resources of views or tables (procedure output is not supported.)

Audience

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

- Architects.
- Developers.
- CIS Administrators.

Caching Overview

Caching in CIS is the materialization of a resource (either view, table, or stored procedure) and storing of the materialized data in an external database or flat file (similar to the way a Materialized View operates in an Oracle database.) Resources in CIS are configured to store materialized data in one or more tables in a relational data source. Once the basic configuration is done, a number of options exist as to when and/or how often a resource's cache is refreshed. However, this refresh is a full refresh of the entire data set every time a refresh is run (meaning an entirely new result set is loaded into the cache data table, after which old result sets from any previous refreshes are removed.)

Full Caching

Full caching is a method of updating all the cache data with the current set of rows from the source(s) of record. Each time the refresh is done it pulls all of the data. This works efficiently for relatively small data sets. It also works the best when caching complex views in the Business or Application layer of the Data Abstraction best practices as those views typically do not yield multi-million row tables. Additionally due to complexities of joins it may not be possible to implement an incremental cache strategy due to conflicting keys.

Incremental Caching

Incremental caching is a method of updating existing cache data with new, updated, or deleted rows from the source(s) of record.

Push-based incremental caching involves being notified of changes by an application or change data capture mechanism and updating the cache data as source data changes. The requirements and setup for this within CIS are described in Appendix F of the CIS User's Guide.

Pull-based incremental caching (which is the implementation that this document discusses) is more of a batch-oriented process where a refresh is initiated in CIS and a mechanism detects what data has changed since the last cache refresh (full or incremental) and applies those changes to the cache data as a single transaction.

There are a number of ways to detect change in data, including scraping database transaction logs, attaching CRUD triggers to database tables, and using a combination of columns to indicate when a row was last inserted, updated or deleted (deleted row data may also be stored in a separate table altogether.)

In this solution, two methods of detecting data change are supported "out of the box":

- One or more "last updated" columns and, for those tables that track it, a separate table of deleted rows to indicate when data was deleted. Also, a way of uniquely identifying each row in the cache data table (with either an actual or logical primary key) is required.
- A separate “delta” table is maintained by the updating application indicating what changes have taken place in the underlying data. This table contains only one record for each corresponding record in the underlying data with an extra column indicating what the last operation was. For example, when a delta row is inserted, the last operation is indicated by a value of ‘C’. When the underlying data is updated, the delta row is updated (instead of a second row being inserted) with the last operation changed to ‘U’. Again, a way of uniquely identifying each row in the cache data table is required.

Either of these methods may be altered / updated to fit other types of data change detection.

Partitioned Incremental Caching

[Not currently implemented].

Partitioned incremental caching is similar to incremental caching, but instead of caching the entire data set, only part of the data is cached and the rest of the data is accessed from the original system(s) of record on demand.

As an example, the cost to cache over 100 years of data cannot be justified when older data will be infrequently accessed. For this reason, a view of such data can be partitioned in such a way that only the most recent data is cached. To implement partitioned caching, some additional components are needed to allow a user to seamlessly and transparently query data from both halves of the partitioned data.

In the current solution, the partitioning is accomplished using a named "activity date" column and the newer data is cached. With some tweaking of the code, this behavior can be altered to fit other partitioning needs.

Caching Data Source Targets

The cache management framework currently supports the following cache target databases:

- Oracle 10g, 11g
- SQL Server 2012
- Netezza 6, 7 [Merge not supported therefore only inserts are supported.]
- Other databases can be supported by engaging a Cisco Advanced Services engineer.

Caching Terms

1) Cache Target – The cache target is one of the supported cache databases.
2) Application – The application refers to the business line, business area or subject area in which the cache framework will be applied. There is one cache target database per application.
3) Context – All procedures within the cache framework are executed using the context of the “constants” procedure which defines the application paths and cache database to be used.
    a. Additionally the context is important from a deployment perspective as the CACHING_DATA table is used to track the views to be cached for a given application.
    b. All audit log entries are saved with the context of the organization name and application name.

Business Requirements

The primary requirement is to create a virtual data model using Composite DV layer. This data model federates and virtualizes data from various data sources servicing specific business areas.

As part of the primary requirements, the virtual model should be able to provide historical information of various source tables. Since all sources of data do not store history, therefore history should to be maintained in DV layer.

Therefore, the caching framework should be able to configure caching on resources

- that are cached in traditional method to improve performance, where traditional approach requires overwrite complete record sets for each cache cycle, or
- configure resources that are cached incrementally to build history.

Technical Requirements

1) Provide a generic cache framework that supports Oracle, SQL Server and Netezza for cache targets
2) Support the following cache types for code generation
    a. ***Incremental*** (no staging) – incrementally acquire newly inserted rows.
    b. ***Full single table*** (uses cachekey) – refresh full cache to a single table.
    c. ***Full multi-table*** (no cachekey) – refresh full cache to a round-robin of multiple tables.
    d. ***Hybrid*** (incremental + staging)
    e. ***Merge*** – ability to merge inserts/updates/deletes from source into incremental cache.
    f. ***Rolling window*** – Purges data that falls outside the window. Need to define purge window and frequency of purge.
    g. ***Partition*** – ability to reference a cache and live partition at the same time.
3) Support the following cache features
    a. Distribution columns for Netezza
    b. Drop/Create indexes
    c. Execute table statistics
    d. Initial/Delta load procedures for incremental refresh.
    e. Pre and Post cache procedure execution for full refresh.
    f. Cache Schedule (currently manual only)
        i. Create cache schedule using period within view
        ii. Create separate trigger to execute "RefreshCache" procedure
        iii. Cache policies can be used with full cache only.
4) Provide a general audit logging framework to correlate messages.
5) Ability to initialize the framework
6) Ability to de-configure a cache resource
7) Ability to deploy a cache resource or all resources in CACHING_DATA

INSTALLATION

Requirements for Installation

- Composite Information Server 6.2 with Service Pack 1 or higher installed.
- AS Development Utilities version 2014Q307 [Utilities_2014Q307.car] installed.
- A caching database must be created, introspected, and configured for caching. This means that:
    a. ***Cache status and cache tracking tables should be created and configured on the cache target data source connection***.
    b. The cache target data source user credentials must have privileges to
        i. Create, drop and get sequences
        ii. Create and drop indexes
        iii. Create and drop tables
        iv. Execute table statistics

Installation Steps

- Create or identify a folder to house the incremental caching code. By default, the solution expects to be installed in the `/shared/ASAssets` folder. When the cache framework solution is imported, the structure `/shared/ASAssets/CacheManagement/CacheFramework` is created. In the rest of this document `<CF_Folder>` will be used to reference this location.

- Import the cache framework solution `CacheFramework-YYYY-MM-DD.car` file into the CIS instance into `/shared/ASAssets/CacheManagement`.

- Import the cache framework virtual database by right clicking on /services/databases and import `CacheFrameworkDB-YYYY-MM-DD.car.`

- (Optional) Import the sample `CacheFrameworkSample-YYYY-MM-DD.car` file into the CIS instance into `/shared/ASAssets/CacheManagement`. When the cache framework sample is imported, the structure `/shared/ASAssets/CacheManagement/CacheFrameworkSample` is created. In the rest of this document `<CS_Folder>` will be used to reference this location.

- Set privileges

    - Set privileges on `/shared/ASAssets/CacheManagement` for composite/all as READ EXECUTE SELECT and push down recursively as "make child resources look like this resource".

- Set privileges on `/services/databases/ASAssets/CacheManagement` for composite/all as READ EXECUTE SELECT and push down recursively as "make child resources look like this resource".

- Initialize the administrative interface database. This is a CIS internal data source that is used to point back to CIS. Open the data source located at `<CF_Folder>/Scripts/AdminInterface/CIS_Internal` and modify the connection parameters to point back to the CIS server that the scripts are installed on. Use an admin user who has rights to execute and copy privileges.
- Proceed to "Configuring an Application Template" to configuring the cache scripts.
- Proceed to "Configuring the Cache Framework Sample" for configuring the sample.
- If not using one of the supported databases as a caching database, please contact your account manager to engage the services of an AS engineer who may be able to update the solution to use your caching database type.

CONFIGURE CACHE FRAMEWORK

The following is a discussion on how to configure a view using the cache framework.

Introduction

The concept of porting an application template is to provide the ability to generate and deploy CIS cache objects. The application template provides all of the necessary hooks into the generic framework code for implementing and deploying cache resources. It is important to understand that the application sets the context. There are different perspectives that can be considered when defining the context.

1) *Subject Area context* – Conceptually, this is the better choice irrespective of whether there is one cache database or multiple. By having an application configured for each subject area, it allows the audit log messages to be customized with same "*Organization Name*" and "*Application Name*". In this scenario, the application will only see their audit log messages because the other messages are filtered out. It also allows the application users to control their own list of cache resources separate from other applications within the CACHING_DATA table.
2) *Target cache database context* – Technically speaking only one context is required for each cache database being referenced in CIS. This is true because the paths to the cache database are referenced in the "Constants" procedure. However, this also means that all audit log messages will have the same Organization and Application Name and the list of cache resources will be within a single context thus there is no way to separate out the notion of different subjects.
3) It is strongly advised that the view to be cache have a primary key index created on it (that includes the cache key column.) In any case, a column or combination of columns is required in order to uniquely identify a row of cached data.
4) If the "last updated" type of incremental caching is going to be used, then an index on the "last updated" column(s) would be advisable as well.

Configuring an Application Template

Configuring a template is a one-time action that should be done by a CIS architect with overall knowledge of the application and cache database target.

1) Copy the application template applicable for your cache database type:
    a. Oracle: `<CF_Folder>`/ApplicationTemplateOracle
    b. SQL Server: `<CF_Folder>`/ApplicationTemplateSqlServer
    c. Netezza: `<CF_Folder>`/ApplicationTemplateNetezza
    d. Copy and paste the folder into your own application folder.
        i. Resources will automatically rebind to your folder location.

    e. Rename the target “TG” folder to meet your needs which shall be known as `<TG_Folder>` throughout this document.

2) Modify constants

    a. Location: `<TG_Folder>`/<ApplicationTemplate>/Constants/***constants***

    b. Minimum constants to modify:

        i. ***ApplicationBasePathConstant*** – the new location of your application template folder.

        ii. ***DEBUG_LEVEL*** – Set for DEBUG_LEVEL_INFO but it is recommended to set it to DEBUG_LEVEL_DEBUG until you are comfortable with how everything is working.

        iii. ***DEBUG_LOGGING_TYPE*** – Set to save to DB (AUDIT_LOG) and PRINT to the command line. Remove PRINT to disable printing to command line.

        iv. ***OrganizationName*** – The organization such as “Mortgage”

        v. ***ApplicationName*** – Your application name. Recommend staying with architectural concepts such as Business Line/Business Area/Subject Area so that audit log messages can be correlated with folders in CIS. Functional concept.

        vi. ***DefaultCacheDSPath*** – The CIS path to the cache data source

        vii. ***DefaultCacheDSCatalogName*** – The name of the catalog if applicable or NULL.

        viii. ***DefaultCacheDSSchemaName*** – The name of the schema.

        ix. All other variables are optional and can be configured to the users liking.

3) Rebind the packaged queries to your cache data source

    a. Note: This only needs to be done during the initial configuration of the Cache Framework application template scripts.

    b. Location: `<TG_Folder>`/<ApplicationTemplate>/Initialize

    c. ***RebindPackagedQueries*** – Execute to rebind the two packaged queries found in /PackageQueryDDLScripts. There are no input parameters. The information for rebinding is contained within the Constants procedure.

        i. ExecuteDDL

        ii. GetIntResult

4) Initialize database tables

    a. ***<u>CAUTION</u>***:

        i. This only needs to be done during the initial configuration of the Cache Framework application scripts.

        ii. If multiple “application” folders are sharing the same cache database, then this initialization step only has to be done once for the configuration of the first application. Any subsequent execution of these scripts will result in dropping and recreating existing database objects and thus “DELETING” other application metadata.

   b. Location: `<TG_Folder>`/<ApplicationTemplate>/Initialize
   c. ***CreateDBSequence*** : Create DB sequence CACHE_FRAMEWORK_SEQ. Does not drop automatically.
   d. ***Create_AUDIT_LOG*** : Will automatically drop if exists.
   e. ***Create_AUDIT_LOG_SEQ*** : Will automatically drop if exists.
   f. ***Create_CACHING_DATA*** : Will automatically drop if exists.
   g. Database privileges required:
      i. Create/Drop Sequence
      ii. Create/Drop Table
      iii. Create/Drop Index
   h. About DROP – Use these if you want to decommission the Cache Framework entirely from a target cache database.
      i. Drop_CACHING_DATA : Drop CACHING_DATA table
      ii. Drop_AUDIT_LOG_SEQ : Drop AUDIT_LOG_SEQ table
      iii. Drop_AUDIT_LOG : Drop AUDIT_LOG table
      iv. DropDBSequence : Drop the DB sequence CACHE_FRAMEWORK_SEQ

5) Modify CachingData procedure
   a. Location: `<TG_Folder>`/<ApplicationTemplate>/DataScripts
   b. Configure the list of views to be cached by editing the ***CachingData*** procedure.
      i. Details on configuration settings can be found in the header of the “CachingData” procedure or in the section of the document: “Cache Framework Features and Attributes”.
   c. ***CachingData_Insert*** – Execute to load rows into CACHING_DATA table.
      i. Parameter: 0=Delete and insert only the rows found in CachingData procedure. 1=Delete all rows and then insert CachingData. In both cases, the delete and insert is done in the context of Organization Name and Application Name.

6) Configure the Cache
   a. Location: `<TG_Folder>`/<ApplicationTemplate>/Execute
   b. ***ConfigureCache*** – Use in conjunction with CACHING_DATA table.
      i. Leave inResourcePath blank to configure all views specified in CACHING_DATA or enter a specific resource path to configure only that one.

        ii. Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".

    c. ***ConfigureCacheAdhoc*** – Use when needing to configure a cache in an adhoc way and don't use CACHING_DATA table.

        i. Must supply all parameters for this procedure that would normally be supplied via CACHING_DATA.

7) View of AUDIT_LOG

    a. Location: `<TG_Folder>`/<ApplicationTemplate>/Execute

    b. ***AuditLogDisplay*** – Displays the results of AUDIT_LOG for your "*Organization Name*" and "*Application Name*" in descending order or so the latest message is displayed first. It filters out other organizations and applications.

8) Refresh the cache

    a. Location: `<TG_Folder>`/<ApplicationTemplate>/Execute

    b. ***RefreshCache*** – Enter the path of the view to refresh.

    c. Run AuditLogDisplay to view AUDIT_LOG

9) De-configure the cache

    a. Location: `<TG_Folder>`/<ApplicationTemplate>/Execute

    b. ***Deconfigure*** – De-configure a cache or list of cache resources found in CACHING_DATA.

        i. Enter the path of the view to de-configure.

        ii. If left NULL, then de-configure all cache resources found in CACHING_DATA filtered by Organization Name and Application Name.

        iii. It is recommended to test that this works on one configured view.

    c. Run AuditLogDisplay to view AUDIT_LOG

CACHE FRAMEWORK FEATURES

The following is a discussion on cache framework features.

Log to a generic audit table script

1) Supported and tested for all cache database targets.
2) cfLog – the main logging procedure invoked by all cache framework procedures. This procedure provides the framework for the various debug levels. This procedure is an interface procedure and does not implement the actual logging itself.
3) The generic "auditLogger" script is the implementation for the different debug logging types.
4) Debug Levels implemented
    a. Various procedures in the Cache Framework implement different levels as shown below. The user can control which levels get logged simply by setting DEBUG_LEVEL in the Constants file.
        i. DEBUG_LEVEL_ERROR=0 - Output error message to log file with severity level ERROR. No command line print.
        ii. DEBUG_LEVEL_AUDIT=1 - Output audit message to log file with severity level INFO. No command line print.
        iii. DEBUG_LEVEL_INFO=2 - Output info message to log file with severity level INFO. No command line print.
        iv. DEBUG_LEVEL_DEBUG=3 - Output debug message to log file with severity level INFO. No command line print.
    b. The user can control the debug logging type which specifies where to write the message to. The user can set DEBUG_LOGGING_TYPE to one or more options include [LOG, DB, PRINT, EMAIL].
        i. LOG – outputs message to the CIS log file cs_server.log.
        ii. DB – outputs message to the AUDIT_LOG table within the cache database that is supported.
        iii. PRINT – outputs message to the Studio command line. This is useful for manual execution and debugging.
        iv. EMAIL – outputs message to an email. Must configure email in the CIS Studio Administration, Configuration: /Composite Server/Configuration/E-Mail.

Cache Types

1) "FS" = Full Single table cache
2) "FM" = Full Multi-table cache

3) "IN" = Incremental cache (no staging)
4) Hybrid (staging + incremental)
    a. "HS" = Hybrid with Staging for Initial and Delta (same script)
        i. HS = **H**ybrid Delta **S**tage
    b. "HN" = Hybrid with Staging for Initial and NO staging for Delta (different scripts)
        i. HN = **H**ybrid Delta **N**o-Stage
    c. "MT1" = Merge Type 1 (Staging for initial script and staging for delta script)
        i. MT1=**M**erge **T**ype **1**
    d. "MT2" = Merge Type 1 (Staging for initial script and staging for delta script)
        i. MT2=**M**erge **T**ype **2**
    e. "MT4" = Merge Type 1 (Staging for initial script and staging for delta script)
        i. MT4=**M**erge **T**ype **4**
5) Not implemented
    a. Partition – ability to have some data cached and some data live via a UNION

Cache Features and Attributes

1) **Distribution Column** support for Netezza – supported
    a. CONFIGURE: Netezza Distribution Column Attribute
    b. **DISTRIBUTION_COLUMNS**: Column(s) that can be used to create distribution key on NZ.
        i. <attribute><name>DISTRIBUTION_COLUMNS</name> <value>NULL</value></attribute>
2) **Multi-Table configuration attributes**
    a. CONFIGURE: Full Multi-Table Caching Attributes:
    b. **MULTI_BUCKET_MODE**: AUTO_GEN or MANUAL. Only AUTO_GEN is supported at this time.
        i. <attribute><name>MULTI_BUCKET_MODE</name> <value>AUTO_GEN</value></attribute>
    c. **MULTI_BUCKET_TABLE_PREFIX**: Provide an explicit name or the $DEFAULT_RES_NAME is extracted dynamically from the resource being cached.
        i. <attribute><name>MULTI_BUCKET_TABLE_PREFIX</name> <value>$DEFAULT_RES_NAME</value></attribute>

d. **MULTI_NUM_BUCKETS**: Default is 3. The number of cache table buckets to create by appending 0, 1, 2... to the end of the bucket table prefix.

    i. <attribute><name>MULTI_NUM_BUCKETS</name><value>3</value></attribute>

e. **MULTI_DROP_CREATE_INDEX**: true/false to drop and create any indexes associated with the cache view.

    i. <attribute><name>MULTI_DROP_CREATE_INDEX</name><value>true</value></attribute>

3) **Incremental and Hybrid Load script configuration**

    a. Name type – format:

        i. "HS" – VIEWNAME_CacheLoad

        ii. "HN" – VIEWNAME_CacheInitialLoad

        iii. "HN" – VIEWNAME_CacheDeltaLoad

        iv. "IN" – VIEWNAME_CacheInitialLoad

        v. "IN" – VIEWNAME_CacheDeltaLoad

    b. CONFIGURE: Incremental Caching Attributes:

    c. **INCR_KEY_NAME**: Incremental cache column name.

        i. <attribute><name>INCR_KEY_NAME</name><value>UPD</value></attribute>

    d. **INCR_KEY_TYPE**: Incremental cache column data type for the above column.

        i. <attribute><name>INCR_KEY_TYPE</name><value>TIMESTAMP</value></attribute>

    e. **INCR_KEY_STARTING_VALUE**: Incremental cache initial default value if history needs to be loaded.

        i. <attribute><name>INCR_KEY_STARTING_VALUE</name><value>1900-01-01 00:00:00</value></attribute>

4) **Full Cache Pre/Post cache procedure configuration**

    a. Supported for Full and Multi-table

    b. Name format:

        i. VIEWNAME_preCallback – interface procedure

        ii. VIEWNAME_preCallbackImpl – developer modifies this procedure

        iii. VIEWNAME_postCallback – interface procedure

        iv. VIEWNAME_PostCallbackImpl – developer modifies this procedure

    c. Implementation procedure variables passed in:

        i. constantsPath
        ii. cacheViewPath
        iii. cacheViewName
        iv. cacheKey

    d. CONFIGURE: Full Cache Pre/Post-Refresh Cache Procedures:
    e. **FULL_PRE_CALLBACK_IMPLE_PROC**: Explicit path to the pre-refresh callback implementation procedure or use the default application cache proc path defined in CONSTANTS along with a generated procedure name.
        i. <attribute><name>FULL_PRE_CALLBACK_IMPL_PROC</name><value>$APPLICATION_CACHE_PROC_IMPL</value></attribute>
    f. **FULL_POST_CALLBACK_IMPL_PROC**: Explicit path to the post-refresh callback implemenation procedure or use the default application cache proc path defined in CONSTANTS along with a generated procedure name.
        i. <attribute><name>FULL_POST_CALLBACK_IMPL_PROC</name><value>$APPLICATION_CACHE_PROC_IMPL</value></attribute>

5) Drop/Create indexes

a. Indexes integrated into View “Indexes” tab. If present, indexes are created otherwise no action.

b. Cache table index name format:
    i. Option 1: UPPER(VIEW_NAME)_<actual_idx_name>
    ii. Option 2: INDEX_PREFIX<actual_idx_name>
        1. This uses the INDEX_PREFIX attribute to override the default upper view name with underscore format.

c. CONFIGURE: Index Attributes

d. **INDEX_PREFIX**: Index prefix overrides the use of the upper case cached view name for prefixing the index name found on the index tab of the cached view.
    i. <attribute><name>INDEX_PREFIX</name><value>IDX_</value></attribute>

e. **INDEX_TYPE**: Index type is appended after the indexNameURL in the CREATE INDEX syntax and is database specific. Syntax: CREATE INDEX <indexNameURL> [INDEX_TYPE] ON <tableURL>(<columnList>) [INDEX_OPTIONS]
    i. <attribute><name>INDEX_TYPE</name><value></value></attribute>

f. **INDEX_OPTIONS**: Index options are appended to the end of the CREATE INDEX syntax and are database specific. Syntax: CREATE INDEX

<indexNameURL> [INDEX_TYPE] ON <tableURL>(<columnList>) [INDEX_OPTIONS]

    i. <attribute><name>INDEX_OPTIONS</name><value></value></attribute>

g. For multi-table if the "Drop indexes before load and create indexes after refresh load" box is checked then drop and recreate is done by CIS otherwise skipped.

    i. The attributes REFRESH_DROP_INDEX and REFRESH_CREATE_INDEX are ignored.

h. The attribute **REFRESH_DROP_INDEX** controls whether the indexes should be dropped during refresh. . If not specified no action is taken on indexes during refresh.

    i. The value "ALL" specifies to drop all indexes that are configured in the view. Otherwise, a comma separate list of index names may be provided.

    ii. **REFRESH_DROP_INDEX**: Drop all indexes for the cache table. [ALL, <comma separated list of indexes>]

        1. <attribute><name>REFRESH_DROP_INDEX</name> <value>ALL</value></attribute>

i. The attribute **REFRESH_CREATE_INDEX** controls whether the indexes should be created during refresh. If not specified no action is taken on indexes during refresh.

    i. The value "ALL" specifies to create all indexes that are configured in the view. Otherwise, a comma separate list of index names may be provided.

    ii. **REFRESH_CREATE_INDEX**: Create all indexes for the cache table. [ALL, <comma separated list of indexes>]

        1. <attribute><name>REFRESH_CREATE_INDEX</name> <value>ALL</value></attribute>

6) Execute table statistics

a. The attribute EXECUTE_STATISTICS controls whether to update statistics or not after the refresh cache is complete.

    i. The value "TABLE" and performs an update statistics on the table.

    ii. The value "EXPRESS" is for Netezza only and performs a "generate express statistics" on the table.

b. Note: For Oracle, executing statistics on the table requires a lock on the table.

7) Create Cache Schedule

a. Create cache schedule using period within the cached view.

b. REFRESH: Schedule Attributes

c. **REFRESH_MODE**: Schedule refresh mode [MANUAL, SCHEDULED]

    i. <attribute><name>REFRESH_MODE</name><value>SCHEDULED</value></attribute>

d. **SCHEDULE_MODE**: When refresh mode=SCHEDULED then [CALENDAR]

    i. <attribute><name>SCHEDULE_MODE</name><value>CALENDAR</value></attribute>

e. **SCHEDULE_START_TIME**: When refresh mode=SCHEDULED then a valid timestamp.

    i. <attribute><name>SCHEDULE_START_TIME</name><value>2014-06-25 00:00:00</value></attribute>

f. **SCHEDULE_PERIOD**: When refresh mode=SCHEDULED then values: SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, or YEAR.

    i. <attribute><name>SCHEDULE_PERIOD</name> <value>HOUR</value></attribute>

g. **SCHEDULE_COUNT**: When refresh mode=SCHEDULED then the number of Period units in the interval between cache refreshes. Values: Any positive integer.

    i. <attribute><name>SCHEDULE_COUNT</name><value>1</value></attribute>

h. **EXPERIATION_PERIOD**: The amount of time in milliseconds that the cache will be cleared after it is refreshed. Values: If less than zero, the period will be set to zero. If zero then the cache will never expire. If set to NULL, the enable setting will be left unaltered.

    i. <attribute><name>EXPERIATION_PERIOD</name> <value>1</value></attribute>

i. **CLEAR_RULE**: Indicates when old cache data should be cleared. Values: One of "NONE", "ON_LOAD", or "ON_FAILURE". Normally old cache data is cleared on expiration and when a cache refresh successfully completes. In the latter case the old cache data is replaced by the new cached data. If "NONE", then the normal behavior will be used. If "ON_LOAD", in addition to the normal behavior the old cache data will be cleared when a refresh is started. If "ON_FAILURE", in addition to the normal behavior the old cache data will be cleared when a refresh fails. If set to NULL, the setting will be left unaltered.

    i. <attribute><name>CLEAR_RULE</name><value>NONE</value></attribute>

8) Rolling Purge window

    a. Rolling Window - Data that rolls out of the window is purged

    b. Purge Trigger Schedule Attributes

c. **PURGE_WINDOW_PERIOD**: defines the purge window that gets passed into the PurgeCacheData() procedure. (delete all rows WHERE PURGE_COLUMN_NAME <= CURRENT_TIMESTAMP - (PURGE_WINDOW_COUNT * PURGE_WINDOW_PERIOD)) values: MINUTE, HOUR, DAY, WEEK, MONTH, or YEAR.

    i. <attribute><name>PURGE_WINDOW_PERIOD</name> <value>WEEK</value></attribute>

d. **PURGE_WINDOW_COUNT**: the number of period units in for the purge window and the value passed into PurgeCacheData() procedure. Values: Any positive integer.

    i. <attribute><name>PURGE_WINDOW_COUNT</name> <value>1</value></attribute>

e. **PURGE_START_TIME**: a valid starting timestamp.

    i. <attribute><name>PURGE_START_TIME</name> <value>2014-06-25 00:00:00</value></attribute>

f. **PURGE_COLUMN_NAME**: the timestamp column used for purging rows.

    i. <attribute><name>PURGE_COLUMN_NAME</name> <value>SDT</value></attribute>

g. **PURGE_PERIOD**: defines the trigger interval period between purges. (how often the trigger fires) values: MINUTE, HOUR, DAY, WEEK, MONTH, or YEAR.

    i. <attribute><name>PURGE_PERIOD</name><value>DAY</value></attribute>

h. **PURGE_COUNT**: the number of Period units in the interval between purges. Values: Any positive integer.

    i. <attribute><name>PURGE_COUNT</name><value>1</value></attribute>

i. **PURGE_RECURRING_DAY**: NULL or a list of one or more items: [SUN, MON, TUE, WED, THU, FRI, SAT]. May be space or comma delimited.

    i. <attribute><name>PURGE_RECURRING_DAY</name> <value>MON WED FRI</value></attribute>

j. **PURGE_FROM_TIME_IN_A_DAY**: NULL or The time of day to start the TRIGGER: between 00:00:00 and 23:30 in 30 min increments. Format = HH24:MM:SS

    i. <attribute><name>PURGE_FROM_TIME_IN_A_DAY</name> <value>06:00:00</value></attribute>

k. **PURGE_END_TIME_IN_A_DAY**: NULL or The time of day to end the TRIGGER: between 00:00:00 and 23:30 in 30 min increments. Format = HH24:MM:SS

    i. <attribute><name>PURGE_END_TIME_IN_A_DAY</name> <value>23:59:00</value></attribute>

l. **PURGE_IS_CLUSTER**: NULL or 0=not once per cluster, 1=only once per cluster

    i. <attribute><name>PURGE_IS_CLUSTER</name> <value>1</value></attribute>

m. **PURGE_ANNOTATION**: NULL or an annotation for the trigger.

i. <attribute><name>PURGE_ANNOTATION</name><value>annotation</value></attribute>

9) **Reset maintenance level to force a full refresh**

   a. Reset the maintenance level back to null and reload the CIS cache with the maintenance level.

10) **Validation**

   a. Validate database table and index name length for max allowed

   b. Alleviate naming collisions. Validate the cache or staging view is a dependent of the target DB cache or staging table to insure that the table will not be dropped/created by mistake if the table name actually belongs to a different CIS resource.

Ability to Initialize Framework

1) Rebind packaged queries after installation using the RebindPackagedQueries script.
2) Manage [drop/create] audit log correlation database sequence id CACHE_FRAMEWORK_SEQ
   a. CreateDBSequence
   b. DropDBSequence
3) Manage [drop/create] AUDIT_LOG_SEQ_table
   a. Create_AUDIT_LOG_SEQ
   b. Drop_AUDIT_LOG_SEQ
4) Manage [drop/create] AUDIT_LOG table
   a. Create_AUDIT_LOG
   b. Drop_AUDIT_LOG
5) Manage [drop/create] CACHING_DATA table
   a. Create_CACHING_DATA
   b. Drop_CACHING_DATA
   c. Insert_CACHING_DATA

Ability to De-configure a Cache

The Cache Framework provides the ability to de-configure the cache resource in two different ways:

1. De-configure a single cache resource by providing the path.
2. De-configure all caches in CACHING_DATA by leaving path null.

The following provides a detailed background on what the de-configure operation will execute:

1) Disable primary cache view (disable cache)
2) If Staging table used
    a. Destroy staging view
    b. Destroy CIS staging table
    c. Drop staging table in database
    d. Delete the staging table cache_status row
3) Destroy CIS cache load or callback scripts. Optional to destroy custom callback implementation scripts.
    a. Destroy first call back script [Initial Load or Pre-Callback and Pre-Callback Implementation (optional)]
    b. Destroy second call back script [Initial Load or Pre-Callback and Pre-Callback Implementation (optional)]
4) Destroy CIS cache table
5) Drop cache tables in database
6) De-configure primary cache view
    a. Destroy cache configuration
    b. Delete the cache table cache_status row
7) Destroy the cache refresh trigger if it exists
8) Destroy the cache purge trigger if it exists
9) Delete AUDIT_LOG_SEQ entry
10) Verify the De-configuration

CACHE FRAMEWORK IMPLEMENTATION

The following is a discussion of how the Cache Framework has been implemented.

Full Caching Methodology

At its most basic, the full caching solution involves the following steps:

- Gather all the current rows from the system of record.
- Insert the gathered rows into the cache table.
- If native load is available for the cache target and configured in CIS then use it otherwise use JDBC inserts.

Incremental Caching Methodology

At its most basic, the incremental caching solution involves the following steps:

- This only handles new record inserts since the last refresh.
- Gather the current rows that have been inserted since the last full or incremental refresh.
- Insert the gathered rows into the cache table.

Hybrid Incremental Caching Methodology

At its most basic, the hybrid incremental caching solution involves the following steps:

- This only handles new record inserts since the last refresh.
- For first time initial load perform a native load (if configured) into a staging table and once complete, perform a local database insert/select from staging into the cache table.
- For delta loads, gather the current rows that have been inserted since the last full or incremental refresh.
- Insert the gathered rows into the cache table.

Hybrid Incremental Merge Caching Methodology

At its most basic, the incremental merge caching solution involves the following steps:

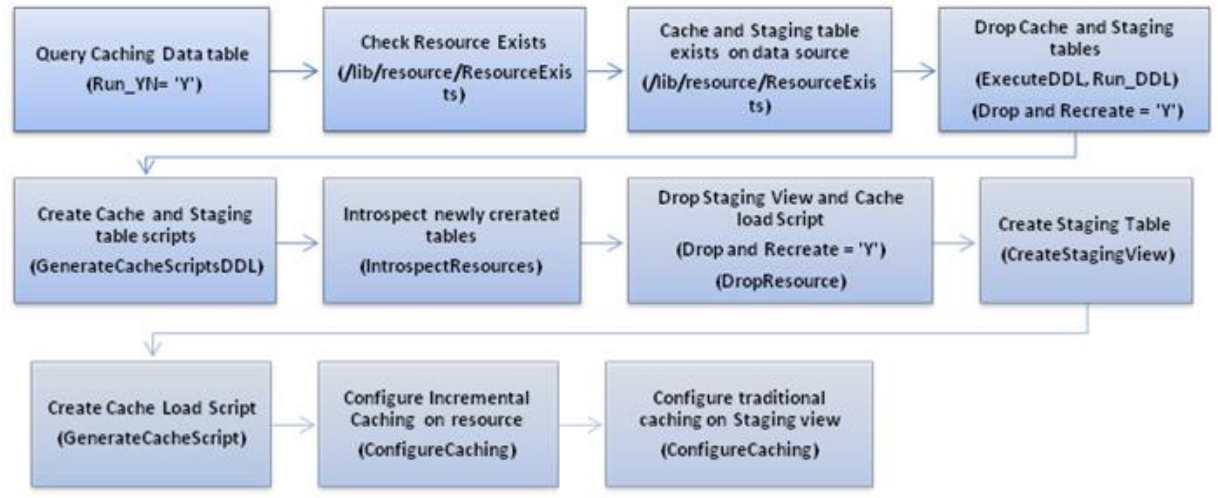
- This only handles inserts, updates and deletes since the last refresh.
- For first time initial load perform a native load (if configured) into a staging table and once complete, perform a local database insert/select from staging into the cache table.

- For delta loads, gather the rows that have been inserted, updated, and deleted since the last full or incremental refresh.
- Insert the gathered rows into a staging table.
- Apply the changed rows to the existing cache data using the caching database's native `MERGE` command. This executes as a single ACID compliant transaction.

Scripts

This folder contains the generic scripts that get invoked by the application template scripts.

The chart below details what the main script <CF_FOLDER>/Scripts/ConfigureCacheScript does when it perform a cache configuration:


Query Caching Data table (Run_YN= 'Y')
Check Resource Exists (/lib/resource/ResourceExists)
Cache and Staging table exists on data source (/lib/resource/ResourceExists)
Drop Cache and Staging tables (ExecuteDDL, Run_DDL) (Drop and Recreate = 'Y')
Create Cache and Staging table scripts (GenerateCacheScriptsDDL)
Introspect newly crerated tables (IntrospectResources)
Drop Staging View and Cache load Script (Drop and Recreate = 'Y') (DropResource)
Create Staging Table (CreateStagingView)
Create Cache Load Script (GenerateCacheScript)
Configure Incremental Caching on resource (ConfigureCaching)
Configure traditional caching on Staging view (ConfigureCaching)

The target data source for caching should be pre-configured for caching. This means that cache status and cache tracking table must be created and introspected. For best results, the case sensitivity and trailing space setting should match the settings of the view that is cached. For each of the views that need to be cached, the “ConfigureCache” script iterates through the list and performs following steps:

1. Check if the target data source is configured for caching
2. Call /lib/resource/ResourceExists API and check if the view exists.
3. If drop and recreate database tables is Y (DROP_RECREATE_TABLE_YN=Y)
   a. Call GetResource API function with view name as parameter to get columns and data type information and create a DDL string for creating a cache table.
   b. When cache type is incremental, calls GetResource API function with view name as parameter to get columns and data type information and create a DDL string for creating a staging table.
   c. Drop caching tables on data source
   d. Drop staging table on the data source if cache type is Hybrid incremental (HS, HN, HM).
   e. Create the cache table by executing the DDL statement created in step 3a.

    f. For incremental caching, create a staging table creating by executing the DDL statement generated in step 3b.
    g. Introspect the data source to bring in the cache and staging target table.
4. If drop and recreate CIS resources is Y (DROP_RECREATE_VIEW_YN=Y)
    a. Drop Staging view resource if drop resources flag is set to Y
    b. Create a staging view as copy of source view in predefined folder when Hybrid incremental (HS, HN, HM, MT1, MT2, MT4).
    c. Drop caching scripts if exist
    d. Generate caching scripts
    e. Configure caching parameters for the view. This step updates the following on the view
        i. Target database
        ii. Target table
        iii. Refresh schedule configure as per attributes
        iv. Update incremental scripts
    f. Configure caching parameters for staging view using default caching mechanism and update the following information
        i. Target database
        ii. Target table for staging view cache
        iii. Manual refresh schedule

NOTE: The caching framework described in above section only configures a resource for caching. The framework does not start a refresh of data.

Cache Framework Application Template Folder

This folder contains the "interface" scripts used to initialize, manage, configure or deconfigure caching for a view. The following sections describe the sub-folders of `<CF_Folder>`/ApplicationTemplate[Oracle|SqlServer|Netezza]. This folder will be referred to as "**ApplicationTemplate**".

ApplicationTemplate/Constants/Constants()

This procedure contains a number of constants and other resources that are used by a number of caching resources. Update the `ApplicationBasePathConstant` variable with the correct value for `<CF_Folder>`.

ApplicationTemplate/DataScripts/CachingData()

This procedure is used to hold data for the caching framework. This script returns a cursor output which is used by ConfigureCache to orchestrate the scripting. The data populated in this script contains the following structure and correlates one for one with the CACHING_DATA database table:

Column Name	Required	Column Description
RUN_YN	Y	Possible values are 'Y' or 'N'. Identifies rows for which caching script should be executed.

DROP_RECREATE_VIEW_YN	Y	Possible values 'Y' or 'N' If 'Y' is provided, the script will drop and recreate all CIS caching framework related objects. If 'N' is provided, existing objects will not be dropped, but ignored.
DROP_RECREATE_TABLE_YN	Y	Possible values 'Y' or 'N' If 'Y' is provided, the script will drop and recreate all database caching and staging tables on the cache target data source. This value is independent of the Drop and Recreate views. If 'N' is provided, existing objects will not be dropped, but ignored.

CACHE_TYPE	Y	The type of cache to configure.
		FS=Full Single
		FM=Full Multi-table
		IN=Incremental
		HS=Hybrid with Delta Staging table
		HN=Hybrid with Delta No Staging Table
		MT1=Merge Type 1 (Oracle and SQL Server)
		MT2=Merge Type 2 (Oracle and SQL Server)
		MT4=Merge Type 4 (Oracle and SQL Server)

RESOURCE_PATH	Y	Full path of the view being cached.
ORGANIZATION	Y	Organization name is used to identify the high-level usage such as "Mortgage". It is used as a filter for AUDIT_LOG and CACHING_DATA.
APPLICATION_NAME	Y	Application name is used to identify the subject area usage. It is used as a filter for AUDIT_LOG and CACHING_DATA.
CONSTANTS_PATH	Y	The full CIS path to the application constants file. This provides the context for all cache framework operations.
ATTRIBUTES	N	Provides an XML configuration of attributes to configure various features. This value can be NULL. Name/Value pair attributes.

ApplicationTemplate/DataScripts/CachingData_Display()

This procedure is used to display the contents of CACHING_DATA table within the context of this "ORGANIZATION" and "APPLICATION_NAME".

ApplicationTemplate/DataScripts/CachingData_Insert()

This script provides a way of inserting rows from the CachingData() procedure into the CACHING_DATA table. The procedure uses a delete filter on “ORGANIZATION” and “APPLICATION_NAME” when deleting records prior to insert.

ApplicationTemplate/Execute/AuditLogDisplay()

This procedure is used to dynamically display rows in the AUDIT_LOG table. The input parameters are qualified by operators such as EQUAL "=" or LIKE "like". If the input parameter does not have a qualifier then it is assumed to use EQUAL "=" when constructing the where clause. If any input parameters are left NULL/blank they are not used to construct the where clause. The following are a list of input parameters that may be used:

- `SEQUENCE_NUM` – The sequence number that is used to correlate across a group of like messages.
- `RESOURCE_NAME_EQUAL` – A resource name equivalent.
- `RESOURCE_NAME_LIKE` – A resource name like using a wildcard at the end of the string.
- `NOTIFICATION_TYPE` – The notification type [ERROR, AUDIT, INFO, DEBUG].
- `ORIG_USER_NAME` – The user@domain that originated the process.

ApplicationTemplate/Execute/ConfigureCache()

This procedure is the main driver script and will do the following when configuring a cache view.

- Query CachngData driver table for all records with RUN_YN = 'Y'
- Create caching table on data source
- Create staging cache table on data source
- Introspect both tables
- Create staging view resource as a copy of primary view
- Create cache load script
- Configure caching on primary view
- Configure caching on staging view (default caching method)
- Drop and recreate caching table, staging table, caching scripts, staging view as well as trigger

The following are a list of input parameters that may be used:

- `inResourcePath` – The specific view path to be configured or leave null to configure all views in CachingData (procedure) / CACHING_DATA (table). This path acts as a filter for what is already configured in CACHING_DATA so the view must have already been configured in CACHING_DATA. Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".

ApplicationTemplate/Execute/ConfigureCacheAdhoc()

This procedure provides a way of configuring a cache view in an adhoc way bypassing the CachingData() procedure. This script performs the same functions as "ConfigureCache" and is the main driver script and will do the following:

- Query CachngData driver table for all records with RUN_YN = 'Y'
- Create caching table on data source
- Create staging cache table on data source
- Introspect both tables
- Create staging view resource as a copy of primary view
- Create cache load script
- Configure caching on primary view
- Configure caching on staging view (default caching method)
- Drop and recreate caching table, staging table, caching scripts, staging view as well as trigger

The following are a list of input parameters that may be used:

- `inResourcePath` – The specific view path to be configured or leave null to configure all views in CachingData (procedure) / CACHING_DATA (table). This path acts as a filter for what is already configured in CACHING_DATA so the view must have already been configured in CACHING_DATA. Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".
- `inCacheType` – Valid types include the list below. Only specified when inResourcePath is specified.
    - FS=Full Single table cache, Optional Pre/Post callback scripts generated when using "ConfigureCache" procedure.
    - FM=Full Multi-table cache, Optional Pre/Post callback scripts generated when using "ConfigureCache" procedure.
    - IN=Incremental cache (no staging tables), Initial Load and Delta Load scripts generated.
    - HS=Hybrid with Delta Staging table, Initial and Delta use the same load script.
    - HN=Hybrid with Delta No Staging Table, Inital Load and Delta Load scripts generated.
    - MT1=Merge Type 1 using Deleted Table. Hybrid Merge Table with Initial and Delta staging table utilized and separate Initial and Delta load script.
    - MT2=Merge Type 2 using History table with Start and End Date (Bi-temporal)
    - MT4=Merge Type 4 using CDC style where there is an activity column specifying the action [I,U,D]. Hybrid Merge Column with Initial and Delta staging table utilized and separate Initial and Delta load script.

- `inDropRecreateViewYN` – Y/N. Drop and recreate the CIS view resource. Only specified when inResourcePath is specified.
- `inDropRecreateTableYN` – Y/N. Drop and recreate the database table. Only specified when inResourcePath is specified.
- `inAttributes` – XML-based attributes if needed. Leave null if no attributes. Only specified when inResourcePath is specified.

ApplicationTemplate/Execute/DeconfigureCache()

This procedure is the main driver script that will de-configure the cache . It will do the following:

1. Disable primary cache view (disable cache)
2. If Staging table used
   a. 2.a. Destroy staging view
   b. 2.b. Destroy CIS staging table
   c. 2.c. Drop staging table in database
   d. 2.d. Delete the staging table cache_status row
3. Destroy CIS cache load or callback scripts. Optional to destroy custom callback implementation scripts.
   a. 3.a. Destroy first call back script [Initial Load or Pre-Callback and Pre-Callback Impl (optional)]
   b. 3.b. Destroy second call back script [Initial Load or Pre-Callback and Pre-Callback Impl (optional)]
4. Destroy CIS cache table
5. Drop cache tables in database
6. De-configure primary cache view
   a. 6.a. Destroy cache configuration
   b. 6.b. Delete the cache table cache_status row
7. Destroy the cache refresh trigger if it exists
8. Destroy the cache purge trigger if it exists
9. Delete AUDIT_LOG_SEQ entry
10. Verify the De-configuration

The following are a list of input parameters that may be used:

- `inResourcePath` – The specific view path to be configured or leave null to configure all views in CachingData (procedure) / CACHING_DATA (table). This path acts as a filter for what is already configured in CACHING_DATA so the view must have already been configured in CACHING_DATA. Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".

ApplicationTemplate/Execute/DeployCache()

This procedure is the main driver script for deploying cache views and tables. The script performs the following:

1. Execute the CachingData_Insert procedure to insert CachingData() rows into CACHING_DATA table
2. Disable the cache views according to CACHING_DATA query
3. Invoke ConfigureCacheScript to create the tables but don't configure the views
4. Update impacted resources
5. Enable the cache views according to CACHING_DATA query

ApplicationTemplate/Execute/PurgeCacheData()

This procedure is a generic script that is invoked by the purge data trigger to purge data within a rolling period of time. The purge time is determined by the current timestamp - the purge period and compared to a timestamp column within the data. All rows are purged (deleted) that fall outside that rolling window.

- `inSequenceName` – The name of the sequence. Leave null when called manually and a sequence will be generated otherwise it is passed in from the invoking procedure.
- `inOrigUserName` – The original user@domain who started the process and transcends sessions. If not set it will be retrieved from the environment.
- `constantsPath` – Path to the application constants procedure.
- `purgeQuantity` – The quantity for the purgePeriod.
- `purgePeriod` – One of [HOUR, DAY, WEEK, MONTH, YEAR]. Note: MONTH is 31 days.
- `purgeColumnName` – The timestamp column used to compare with the purge period.
- `cacheViewPath` – The path to the CIS cache view.

ApplicationTemplate/Execute/RefreshCache()

This procedure is used to kick off a cache refresh on a resource. Currently the default resource type is set to TABLE. The script takes one parameter "resourcePath" which is the full path to the view. The script parses the view path to extract the view name. The script returns SUCCESS or FAIL as output parameter depending if the cache refresh was successful or failure. Failure reason is written to the cache framework log "cfLog" which determines the log type based on how the "Constants" are configured. The log entry may go to the cs_server.log file, Audit_log, or printed to the command line. More than one option can be selected.

This script uses a blocking strategy, therefore, the script will wait till the refresh either completes or fails. The polling time is configurable and is currently set to 30 sec to test whether the cache is completed or not.

ApplicationTemplate/Initialize/Create_AUDIT_LOG()

This procedure is an initialization script used to create the AUDIT_LOG table. This script only needs to be run once. This script will automatically drop the AUDIT_LOG table if it exists before creating it.

ApplicationTemplate/Initialize/Create_AUDIT_LOG_SEQ()

This procedure is an initialization script used to create the AUDIT_LOG_SEQ table. This script only needs to be run once. This script will automatically drop the AUDIT_LOG_SEQ table if it exists before creating it.

ApplicationTemplate/Initialize/Create_CACHING_DATA()

This procedure is an initialization script used to create the CACHING_DATA table. This script only needs to be run once. This script will automatically drop the CACHING_DATA table if it exists before creating it.

ApplicationTemplate/Initialize/CreateDBSequence()

This procedure is an initialization script used to create the CACHE_FRAMEWORK_SEQ sequence. This script does *not* drop the sequence first. The drop sequence must be invoked separately. The CACHE_FRAMEWORK_SEQ is used to correlate a unique number across CIS sessions and transactions. It is used during all operations of the cache framework including Refresh, Configuration, De-configuration and other operations. It uses the AUDIT_LOG_SEQ table to store a correlation name and sequence number. For example, if a cache refresh was started by a user which then goes through the admin interface, the user name is preserved along with the sequence number even though control was transferred to another user. Additionally, the sequence number is preserved through the cache execution of either incremental load scripts or call back scripts.

ApplicationTemplate/Initialize/Drop_AUDIT_LOG()

This procedure is an initialization script used to drop the AUDIT_LOG table. This script also gets invoked by Create_AUDIT_LOG. This script will drop the AUDIT_LOG table if it exists before creating it.

ApplicationTemplate/Initialize/Drop_AUDIT_LOG_SEQ()

This procedure is an initialization script used to drop the AUDIT_LOG_SEQ table. This script also gets invoked by Create_AUDIT_LOG_SEQ. This script will drop the AUDIT_LOG_SEQ table if it exists before creating it.

ApplicationTemplate/Initialize/Drop_CACHING_DATA()

This procedure is an initialization script used to drop the CACHING_DATA table. This script also gets invoked by Create_CACHING_DATA. This script will drop the CACHING_DATA table if it exists before creating it.

ApplicationTemplate/Initialize/DropDBSequence()

This procedure is an initialization script used to drop the CACHE_FRAMEWORK_SEQ sequence.

ApplicationTemplate/Initialize/RebindPackagedQueries()

This procedure is an initialization script used to rebind the packaged queries from the temporary datasource to the one specified in the Constants procedure.

ApplicationTemplate/PackagedQueryDDLScript/TEMP_DS

This is the temporary data source used only at installation time when a rebind of the packaged queries is performed from the temp data source to the actual data source which is configured in the “constants” procedures.

ApplicationTemplate/PackagedQueryDDLScript/ExecuteDDL()

This is a packaged query that is used to execute any DDL on the cache data source. It is rebound to the actual data source at the time of application configuration using the “/Initialize/RebindPackagedQueries” procedure.

The following are a list of input parameters that may be used:

- `inputSQL` – Take a SQL statement.

ApplicationTemplate/PackagedQueryDDLScript/GetIntResult()

This is a packaged query that is used to execute any DML query that returns an integer value on the cache data source. For example SELECT COUNT(*) FROM x. It is rebound to the actual data source at the time of application configuration using the “/Initialize/RebindPackagedQueries” procedure.

The following are a list of input parameters that may be used:

- `selectCountSQLStatement` – Take a Select Count(*) SQL statement.

Published Procedures

The published procedures are those that can be invoked externally by a 3rd party JDBC or ODBC tool.

/databases/ASAssets/CacheManagement/CacheFramework/ClearPlanCache()

This procedure is designed to clear the plan cache. This script uses the admin interface providing elevated privileges to perform this action.

/databases/ASAssets/CacheManagement/CacheFramework/ClearRepositoryCache()

This procedure is designed to clear the repository cache. This script uses the admin interface providing elevated privileges to perform this action.

/databases/ASAssets/CacheManagement/CacheFramework/ConfigureCache()

This procedure uses the admin interface providing elevated privileges to perform this action. Refer back to ApplicationTemplate/Execute/ConfigureCache for more details.

/databases/ASAssets/CacheManagement/CacheFramework/ConfigureCacheAdhoc()

This procedure provides a way of configuring a cache view in an adhoc way bypassing the CachingData() procedure. This script uses the admin interface providing elevated privileges to perform this action. Refer back to ApplicationTemplate/Execute/ConfigureCacheAdhoc for more details.

/databases/ASAssets/CacheManagement/CacheFramework/CopyPrivilegesFromParent()

This procedure is designed to push privileges to a newly create resource and makes is same as the parent folder in which the resource exists. This script uses the admin interface providing elevated privileges to perform this action.

The script takes a resource name with path and resource type as input parameter and returns SUCCESS or FAIL as output message. Additional logging is done in AUDIT_LOG table as well as in cs_server.log file. This script requires the user executing the script to have GRANT privileges on the source folder.

/databases/ASAssets/CacheManagement/CacheFramework/DeconfigureCache()

This procedure is used to de-configure the cache. Refer back to ApplicationTemplate/Execute/DeconfigureCache for more details.

/databases/ASAssets/CacheManagement/CacheFramework/IntrospectTables()

This procedure is used to provide a consistent and generic interface for introspecting database tables. This script uses the admin interface providing elevated privileges to perform this action.

/databases/ASAssets/CacheManagement/CacheFramework/RefreshCache()

This procedure is used to kick off a cache refresh on a resource. Currently the default resource type is set to TABLE. This script uses the admin interface providing elevated privileges to perform this action. Refer back to ApplicationTemplate/Execute/RefreshCache for more details.

/databases/ASAssets/CacheManagement/CacheFramework/UpdateResourceOwner()

This procedure is used to change resource ownership of a resource. This script can be used to change ownership of a single resource or a folder with multiple resources. By default the ownership is pushed recursively to all the resources in a folder. This script uses the admin interface providing elevated privileges to perform this action.

Executing the library function used in this script requires ACCESS TOOLS, READ ALL RESOURCES, READ ALL USERS role, therefore ideally, this script should be executed by an Admin user.

Admin Interface Procedures

The admin interface procedures serve the purpose of allowing a user to execute a procedure with elevated rights by redirecting the procedure through an admin interface CIS internal data source where the user has admin rights.

`<CF_Folder>`**/Scripts/AdminInterface/ClearPlanCache()**

This procedure is designed to clear the plan cache. This script uses the admin interface providing elevated privileges to perform this action.

`<CF_Folder>`**/Scripts/AdminInterface /ClearRepositoryCache()**

This procedure is designed to clear the repository cache. This script uses the admin interface providing elevated privileges to perform this action.

`<CF_Folder>`**/Scripts/AdminInterface /ConfigureCache()**

This procedure uses the admin interface providing elevated privileges to perform this action. Refer back to ApplicationTemplate/Execute/ConfigureCache for more details.

`<CF_Folder>`**/Scripts/AdminInterface /ConfigureCacheAdhoc()**

This procedure provides a way of configuring a cache view in an adhoc way bypassing the CachingData() procedure. This script uses the admin interface providing elevated privileges to perform this action. Refer back to ApplicationTemplate/Execute/ConfigureCacheAdhoc for more details.

`<CF_Folder>`**/Scripts/AdminInterface /CopyPrivilegesFromParent()**

This procedure is designed to push privileges to a newly create resource and makes is same as the parent folder in which the resource exists. This script uses the admin interface providing elevated privileges to perform this action.

The script takes a resource name with path and resource type as input parameter and returns SUCCESS or FAIL as output message. Additional logging is done in AUDIT_LOG table as well as in cs_server.log file. This script requires the user executing the script to have GRANT privileges on the source folder.

`<CF_Folder>`**/Scripts/AdminInterface /DeconfigureCache()**

This procedure is used to de-configure the cache. Refer back to ApplicationTemplate/Execute/DeconfigureCache for more details.

<CF_Folder>/**Scripts/AdminInterface /IntrospectTables()**

This procedure is used to provide a consistent and generic interface for introspecting database tables. This script uses the admin interface providing elevated privileges to perform this action.

<CF_Folder>/**Scripts/AdminInterface /RefreshCache()**

This procedure is used to kick off a cache refresh on a resource. Currently the default resource type is set to TABLE. This script uses the admin interface providing elevated privileges to perform this action. Refer back to ApplicationTemplate/Execute/RefreshCache for more details.

<CF_Folder>/**Scripts/AdminInterface /UpdateResourceOwner()**

This procedure is used to change resource ownership of a resource. This script can be used to change ownership of a single resource or a folder with multiple resources. By default the ownership is pushed recursively to all the resources in a folder. This script uses the admin interface providing elevated privileges to perform this action.

Executing the library function used in this script requires ACCESS TOOLS, READ ALL RESOURCES, READ ALL USERS role, therefore ideally, this script should be executed by an Admin user.

APPENDIX A – CONFIGURE CACHE FRAMEWORK SAMPLE

The following is a discussion of how to augment the Cache Framework.

Configuring the Cache Framework Sample

Configuring the Cache Framework sample is a one-time action that should be done by a CIS architect with overall knowledge of the application and cache database target. There is a sample for each cache database type.

Background Information

1) Samples include Application Template, Views and System of Record (SOR):
2) Purpose – The purpose of the sample is two-fold.
    a. It provides a readymade data source that can be used for quickly learning and demonstrating the Cache Framework capabilities.
    b. It serves as a base-line for testing the Cache Framework and verifying supported data types.
3) Assumptions: The Cache Framework has been installed at this location `<CS_Folder>`:

    `/shared/ASAssets/CacheManagement/CacheFrameworkSample`
4) Configure Oracle Sample
    a. `<CS_Folder>`/Oracle_Application – Cache Framework template for Oracle.
    b. `<CS_Folder>`/Oracle_SampleViews – Sample views to configure caching with.
    c. `<CS_Folder>`/Oracle_SampleSOR – Sample system of record database and scripts.
5) Configure SQL Server Sample
    a. `<CS_Folder>`/SqlServer_Application – Cache Framework template for SQL Server.
    b. `<CS_Folder>`/SqlServer_SampleViews – Sample views to configure caching with.
    c. `<CS_Folder>`/SqlServer_SampleSOR – Sample system of record database and scripts.
6) Configure Netezza Sample
    a. `<CS_Folder>`/Netezza_Application – Cache Framework template for Netezza.
    b. `<CS_Folder>`/Netezza_SampleViews – Sample views to configure caching with.
    c. `<CS_Folder>`/Netezza_SampleSOR – Sample system of record database and scripts.
7) Configure MySQL SOR Sample
    a. `<CS_Folder>`/MySQL_SampleSOR – Sample system of record database and scripts.

Configure Oracle Sample:

1) Application
    a. Location: `<CS_Folder>`/Oracle_Application

   b. Provides a cache target configured sample. It originated from `<CF_Folder>`/ApplicationTemplateOracle. It points to the Oracle SOR for a cache target data source.

2) Views

   a. Location: `<CS_Folder>`/Oracle_SampleViews

   b. Provides a sample view for each cache type. The CachingData procedure is already configured for this sample for the default location of the Cache Framework Sample. There is also a view that exercises the different data types. These views provide a baseline for testing and examples.

3) SOR

   a. Location: `<CS_Folder>`/Oracle_SampleSOR

   b. The SAMPLE_DS serves as both a system or record and the cache target database to simply the connectivity for this sample.

4) **Configure Oracle system of record samples**

   a. Select an Oracle database for creating a sample system of record

      i. Create a user/schema with the name “CIS_CACHE”

      ii. Grant the user the ability to do the following:

```
-- Provide password and tablespace details
CREATE USER CIS_CACHE IDENTIFIED BY <password> default tablespace ts_users temporary tablespace temp;
GRANT CREATE SESSION TO CIS_CACHE;
GRANT CREATE SEQUENCE TO CIS_CACHE;
GRANT CREATE TABLE TO CIS_CACHE;
GRANT CREATE ANY INDEX TO CIS_CACHE;
GRANT CREATE TRIGGER TO CIS_CACHE;
GRANT DROP ANY SEQUENCE TO CIS_CACHE;
GRANT DROP ANY TABLE TO CIS_CACHE;
GRANT DROP ANY INDEX TO CIS_CACHE;
GRANT DROP ANY TRIGGER TO CIS_CACHE;
GRANT ALTER SESSION TO CIS_CACHE;
COMMIT;
-- Provide password and Database SID
CONNECT CIS_CACHE/<password>@<DB>;
ALTER SESSION SET NLS_LANGUAGE=American;
ALTER SESSION SET NLS_TERRITORY=America;
COMMIT;
```

      iii. Note: if a different schema is provided then the scripts will need to be rebound to that database and user once all of the tables are created and introspected.

   b. Open the SAMPLE_DS data source connection.

        i. Edit the connection details to point to the new
        ii. Click “Test Connection” to verify that the connection is working.
    c. Execute create_TEST_INCR to create the following:
        i. TEST_INCR
        ii. TEST_INCR_DEL
        iii. TEST_INCR_HIST
        iv. TEST_INCR_TRIGGER
        v. TEST_INCR_DEL_TRIGGER
    d. Re-introspect the SAMPLE_DS data source.
    e. Validate the SOR tables.
        i. Right click on ../SOR/TEST_INCR and "Execute" to insure no error is thrown.
        ii. Repeat for TEST_INCR_DEL
        iii. Repeat for TEST_INCR_HIST
    f. Insert/Update/Delete rows as needed to test various cache types:
        i. insert_TEST_INCR
        ii. update_TEST_INCR
        iii. deletePK_TEST_INCR

5) **Configure Oracle Application Template**
    a. RebindPackagedQueries
        i. Location: `<CS_Folder>`/Oracle_Application/Initialize/RebindPackagedQueries
        ii. Execute to rebind the two packaged queries [ExecuteDDL and GetIntResult] to the SAMPLE_DS data source. There are no input parameters. The information for rebinding is contained within the Constants procedure.
    b. Initialize database tables
        i. Location: `<CS_Folder>`/Oracle_Application/Initialize
        ii. ***CreateDBSequence*** : Create DB sequence CACHE_FRAMEWORK_SEQ. Does not drop automatically.
        iii. ***Create_AUDIT_LOG*** : Will automatically drop if exists.
        iv. ***Create_AUDIT_LOG_SEQ*** : Will automatically drop if exists.
        v. ***Create_CACHING_DATA*** : Will automatically drop if exists.
    c. Execute CachingData_Insert procedure
        i. Location: `<CS_Folder>`/Oracle_Application/DataScripts

ii. ***CachingData_Insert(1)*** **–** Execute to load rows into CACHING_DATA table. Provide a 1 to delete existing rows before inserting.

d. **Cache Configuration Generation**

i. **ConfigureCache**

1. Location: `<CS_Folder>`/Oracle_Application/Execute/ConfigureCache
2. Configure cache, scripts, schedule and database tables.
3. Input: The path to view to be configured:

   /shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle_SampleViews/Views/V_TEST_INCR_HYB_N

ii. **RefreshCache**

1. Location: `<CS_Folder>`/Oracle_Application/Execute/RefreshCache
2. Refresh the cache.
3. Input: The path to view to be configured:

   /shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle_SampleViews/Views/V_TEST_INCR_HYB_N

iii. **AuditLogDisplay**

1. Report on the outcome of the cache or configuration.
2. `<CS_Folder>`/Oracle_Application/Execute/AuditLogDisplay
3. Input: leave all blank to select all.

iv. **DeconfigureCache**

1. De-configure cache, scripts and database tables.
2. `<CS_Folder>`/Oracle_Application/Execute/DeconfigureCache
3. Input: The path to view to be configured:

   /shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle_SampleViews/Views/V_TEST_INCR_HYB_N

Configure SQL Server Sample:

1) Application

   a. Location: `<CS_Folder>`/SqlServer_Application

   b. Provides a cache target configured sample. It originated from `<CF_Folder>`/ApplicationTemplateSqlServer. It points to the Sql Server SOR for a cache target data source.

2) Views

   a. Location: `<CS_Folder>`/SqlServer_SampleViews

   b. Provides a sample view for each cache type. The CachingData procedure is already configured for this sample for the default location of the Cache

Framework Sample. There is also a view that exercises the different data types. These views provide a baseline for testing and examples.

3) SOR
   a. Location: `<CS_Folder>`/SqlServer_ampleSOR
   b. The SAMPLE_DS serves as both a system or record and the cache target database to simply the connectivity for this sample.

4) **Configure SQL Server system of record samples**
   a. Select a SQL Server database for creating a sample system of record
      i. Create a catalog with the name “CIS_CACHE” and schema “dbo”. Create a user CIS_CACHE.
      ii. Grant the user the ability to do the following:

```
USE [CIS_CACHE];
GRANT CREATE SEQUENCE ON SCHEMA::dbo TO [CIS_CACHE];
GRANT CREATE TABLE TO [CIS_CACHE];
GRANT ALTER ON SCHEMA::dbo TO [CIS_CACHE];
GO;
```

      iii. Note: if a different catalog and schema is provided then the scripts will need to be rebound to that database and user once all of the tables are created and introspected.
   b. Open the SAMPLE_DS data source connection.
      i. Click “Test Connection” to verify that the connection is working.
   c. Execute create_TEST_INCR to create the following:
      i. TEST_INCR
      ii. TEST_INCR_DEL
      iii. TEST_INCR_HIST
      iv. TEST_INCR_TRIGGER
      v. TEST_INCR_DEL_TRIGGER
   d. Re-introspect the SAMPLE_DS data source.
   e. Validate the SOR tables.
      i. Right click on ../SOR/TEST_INCR and "Execute" to insure no error is thrown.
      ii. Repeat for TEST_INCR_DEL
      iii. Repeat for TEST_INCR_HIST
   f. Insert/Update/Delete rows as needed to test various cache types:
      i. insert_TEST_INCR
      ii. update_TEST_INCR
      iii. deletePK_TEST_INCR

5) **Configure SQL Server Application Template**

   a. RebindPackagedQueries

      i. Location: `<CS_Folder>`/SqlServer_Application/Initialize/RebindPackagedQueries

      ii. Execute to rebind the two packaged queries [ExecuteDDL and GetIntResult] to the SAMPLE_DS data source. There are no input parameters. The information for rebinding is contained within the Constants procedure.

   b. Initialize database tables

      i. Location: `<CS_Folder>`/SqlServer_Application/Initialize

      ii. ***CreateDBSequence*** : Create DB sequence CACHE_FRAMEWORK_SEQ. Does not drop automatically.

      iii. ***Create_AUDIT_LOG*** : Will automatically drop if exists.

      iv. ***Create_AUDIT_LOG_SEQ*** : Will automatically drop if exists.

      v. ***Create_CACHING_DATA*** : Will automatically drop if exists.

   vi. Execute CachingData_Insert procedure

      1. Location: `<CS_Folder>`/SqlServer_Application/DataScripts

      2. ***CachingData_Insert(1)*** **–** Execute to load rows into CACHING_DATA table. Provide a 1 to delete existing rows before inserting.

   c. **Cache Configuration Generation**

      i. **ConfigureCache**

         1. Location: `<CS_Folder>`/SqlServer_Application/Execute/ConfigureCache

         2. Configure cache, scripts, schedule and database tables.

         3. Input: The path to view to be configured:

            /shared/ASAssets/CacheManagement/CacheFrameworkSample/SqlServer_SampleViews/Views/V_TEST_INCR_HYB_N

      ii. **RefreshCache**

         1. Location: `<CS_Folder>`/SqlServer_Application/Execute/RefreshCache

         2. Refresh the cache.

         3. Input: The path to view to be configured:

            /shared/ASAssets/CacheManagement/CacheFrameworkSample/SqlServer_SampleViews/Views/V_TEST_INCR_HYB_N

      iii. **AuditLogDisplay**

         1. Location: `<CS_Folder>`/SqlServer_Application/Execute/AuditLogDisplay

2. Report on the outcome of the cache or configuration.
3. Input: leave all blank to select all.

iv. **DeconfigureCache**

1. Location: `<CS_Folder>`/SqlServer_Application/Execute/DeconfigureCache
2. De-configure cache, scripts and database tables.
3. Input: The path to view to be configured:

   /shared/ASAssets/CacheManagement/CacheFrameworkSample/SqlServer_SampleViews/Views/V_TEST_INCR_HYB_N

Configure Netezza Sample:

1) Application
   a. Location: `<CS_Folder>`/Netezza_Application
   b. Provides a cache target configured sample. It originated from `<CF_Folder>`/ApplicationTemplateNetezza. It points to the Netezza SOR for a cache target data source.
2) Views
   a. Location: `<CS_Folder>`/Netezza_SampleViews
   b. Provides a sample view for each cache type. The CachingData procedure is already configured for this sample for the default location of the Cache Framework Sample. There is also a view that exercises the different data types. These views provide a baseline for testing and examples.
3) SOR
   a. Location: `<CS_Folder>`/Netezza_SampleSOR
   b. The SAMPLE_DS serves as both a system or record and the cache target database to simply the connectivity for this sample.
4) **Configure Netezza system of record samples**
   a. Select a Netezza database for creating a sample system of record
      i. Create a database in the Netezza data source called "CIS_CACHE" owned by the user ADMIN.
      ii. Grant the user the ability to do the following:

```
GRANT LIST ON ADMIN TO ADMIN ;
GRANT CREATE SEQUENCE TO ADMIN ;
GRANT CREATE TABLE TO ADMIN ;
GRANT DROP ON SEQUENCE TO ADMIN ;
GRANT DROP ON TABLE TO ADMIN ;
COMMIT;
```

        iii. Note: if a different database and user is provided then the scripts will need to be rebound to that database and user once all of the tables are created and introspected.

b. Open the SAMPLE_DS data source connection.
    i. Edit the connection details to point to the new
    ii. Click “Test Connection” to verify that the connection is working.
c. Execute create_TEST_INCR to create the following:
    i. TEST_INCR
    ii. TEST_INCR_DEL
    iii. TEST_INCR_HIST
d. Re-introspect the SAMPLE_DS data source.
e. Validate the SOR tables.
    i. Right click on ../SOR/TEST_INCR and "Execute" to insure no error is thrown.
    ii. Repeat for TEST_INCR_DEL
    iii. Repeat for TEST_INCR_HIST
f. Insert/Update/Delete rows as needed to test various cache types:
    i. insert_TEST_INCR
    ii. update_TEST_INCR
    iii. deletePK_TEST_INCR

5) **Configure SQL Server Application Template**
    a. RebindPackagedQueries
        i. Location: `<CS_Folder>`/Netezza_Application/Initialize/RebindPackagedQueries
        ii. Execute to rebind the two packaged queries [ExecuteDDL and GetIntResult] to the SAMPLE_DS data source. There are no input parameters. The information for rebinding is contained within the Constants procedure.
    b. Initialize database tables
        i. Location: `<CS_Folder>`/Netezza_Application/Initialize
        ii. ***CreateDBSequence*** : Create DB sequence CACHE_FRAMEWORK_SEQ. Does not drop automatically.
        iii. ***Create_AUDIT_LOG*** : Will automatically drop if exists.
        iv. ***Create_AUDIT_LOG_SEQ*** : Will automatically drop if exists.
        v. ***Create_CACHING_DATA*** : Will automatically drop if exists.
    c. Execute CachingData_Insert procedure

i. Location: `<CS_Folder>`/Netezza_Application/DataScripts

ii. ***CachingData_Insert(1)* –** Execute to load rows into CACHING_DATA table. Provide a 1 to delete existing rows before inserting.

6) **Cache Configuration Generation**

   a. **ConfigureCache**

      i. Location: `<CS_Folder>`/Netezza_Application/Execute/ConfigureCache

      ii. Configure cache, scripts, schedule and database tables.

      iii. Input: The path to view to be configured:

      /shared/ASAssets/CacheManagement/CacheFrameworkSample/Netezza_SampleViews/Views/V_TEST_INCR_HYB_N

   b. **RefreshCache**

      i. Location: `<CS_Folder>`/Netezza_Application/Execute/RefreshCache

      ii. Refresh the cache.

      iii. Input: The path to view to be configured:

      /shared/ASAssets/CacheManagement/CacheFrameworkSample/Netezza_SampleViews/Views/V_TEST_INCR_HYB_N

   c. **AuditLogDisplay**

      i. Location: `<CS_Folder>`/Netezza_Application/Execute/AuditLogDisplay

      ii. Report on the outcome of the cache or configuration.

      iii. Input: leave all blank to select all.

   d. **DeconfigureCache**

      i. Location: `<CS_Folder>`/Netezza_Application/Execute/DeconfigureCache

      ii. De-configure cache, scripts and database tables.

      iii. Input: The path to view to be configured:

      /shared/ASAssets/CacheManagement/CacheFrameworkSample/Netezza_SampleViews/Views/V_TEST_INCR_HYB_N

Configure MySQL SOR:

1) SOR – `<CS_Folder>`/MySQL_SampleSOR

2) Note – Only contains a sample system of record. MySQL is not supported for a cache target database.

3) **Configure MySQL system of record sample**

   a. The CIS samples must already be installed on the CIS server.

   b. Open the SAMPLE_DS data source connection.

      i. Click “Test Connection” to verify that the connection is working.

c. Execute create_TEST_INCR to create the following:
    i. TEST_INCR
    ii. TEST_INCR_DEL
    iii. TEST_INCR_HIST
d. Proceed to “**Common SOR Configuration**” below.

APPENDIX B – ADDING A NEW CACHE TYPE

The following is a discussion of how to augment the Cache Framework.

Adding a New Cache Type to the Source Code

Adding a new cache type involves the following:

1) To add a new cache type, modify these procedure and create an implementation procedure
   a. Create a new implementation procedure that follows the pattern of one like
      i. GenerateCacheScriptsFullCallback
      ii. GenerateCacheScriptsHybridDeltaNoStage
      iii. GenerateCacheScriptsHybridDeltaStage
      iv. GenerateCacheScriptsIncremental
      v. GenerateCacheScriptsHybridDeltaMergeType1
      vi. GenerateCacheScriptsHybridDeltaMergeType2
      vii. GenerateCacheScriptsHybridDeltaMergeType4
   b. List of procedures to modify
      i. CommonTypes.ValidCacheTypesAll
      ii. ConfigureCacheScript
      iii. DeleteReloadCache
      iv. UpdateCacheConfiguration
      v. PublishedImpl/DeployCache
      vi. HelperScripts/ValidateTableDependency
   c. List of Template procedures to modify
      i. ApplicationTemplateNetezza/DataScripts/CachingData
      ii. ApplicationTemplateNetezza/Execute/ConfigureCacheAdhoc
      iii. ApplicationTemplateOracle/DataScripts/CachingData
      iv. ApplicationTemplateOracle/Execute/ConfigureCacheAdhoc
      v. ApplicationTemplateSqlServer/DataScripts/CachingData
      vi. ApplicationTemplateSqlServer/Execute/ConfigureCacheAdhoc