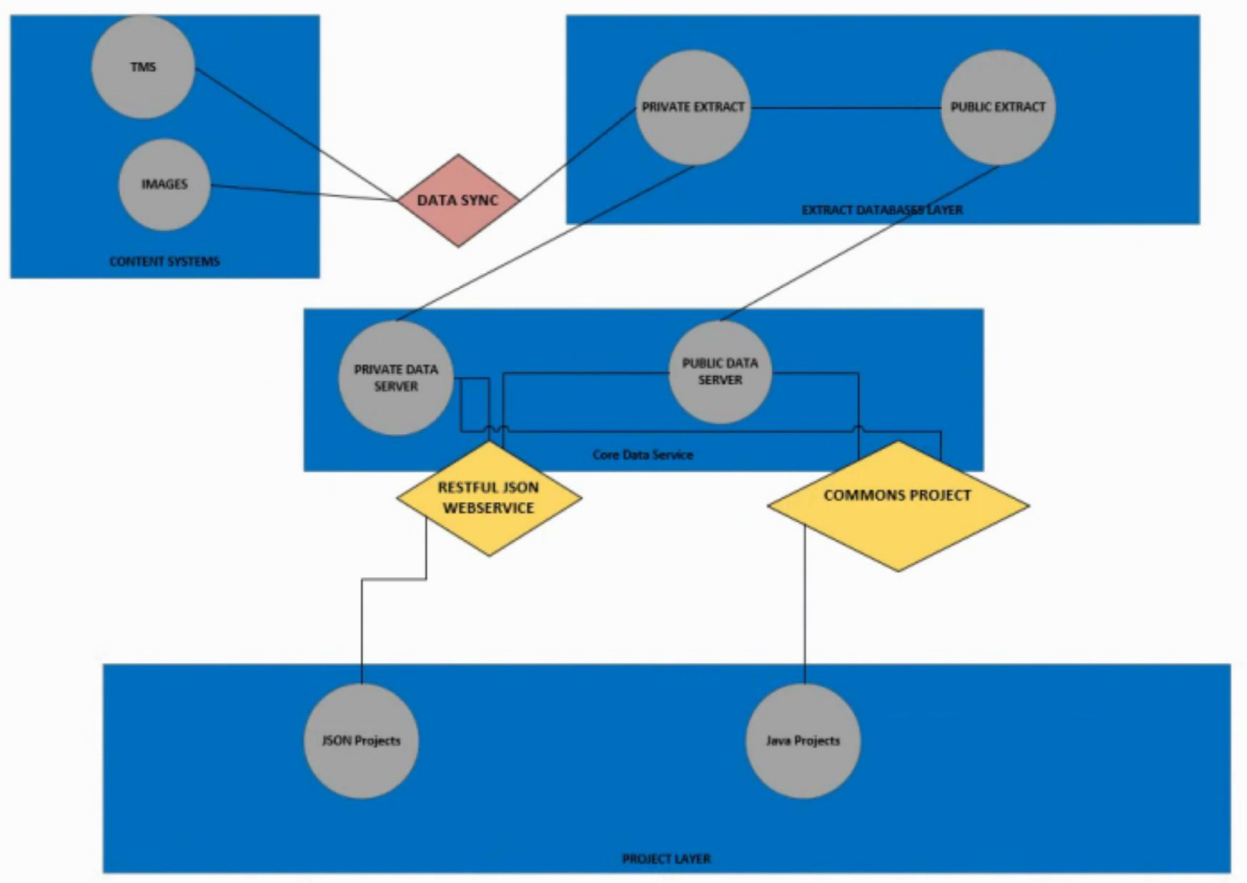
Using Core Data Service via Java APIs in AEM

Overview

TTS has been progressing over the pass few years in centralizing the modeling of the modeling of the TMS data structures. This has resulted in the Java classes the make up the core of the Data Sever, Web Redesign, and Commons API projects. But as the multiple projects mentioned previously would indicate, further work needed to be done in order to both centralize the data modeling and the code base for accessing the actual data.

The next step in this work has been achieved with the Core Data Service (CDS) project. This project brings together the Centralizing of the Data Modeling (Commons Project) with the centralizing of the data repository through support for multiple types of Remote Procedure Calls. This document focuses on an overview of how Core Data Service is used through Java APIs.

Design



The above diagram is an illustration of the current architecture for the Core Date Service. As illustrated, The Commons Project is responsible for defining the Date Models as well as supplying access for Java project retrieving TMS data from the Core Data Service.

The Commons Project

The Commons Project originated as an AEM specific Java project. Starting from Version 2.1, The Commons Project is now a Stand Alone project that produces JAR files as well as bundle files. The bundle dependency is added to a Maven file like below.

```
<dependency>
    <groupId>gov.nga</groupId>
    <artifactId>nga-common</artifactId>
    <version>2.1.0-SNAPSHOT</version>
</dependency>
```

Configuration

Since the Commons Project is now Stand Alone. It is the AEM's project responsibility to deploy it to the environment. This can be done by adding the Commons Project to the list of resources embedded with the "`filevault-package-maven-plugin`" plugin

```
<embedded>
    <groupId>gov.nga</groupId>
    <artifactId>nga-common</artifactId>
    <target>/apps/SimpleClientAEM/install</target>
</embedded>
```

Configuration for CDS is handled by the ClientConfiguration Interface. Each Java project must create its own Object instance of this configuration. Currently, the interface only has 1 method "getConnectionURL()". This is expected to return the URL (including port) for CDS.

```
1  package gov.nga.common.rpc.client;
2
3  public interface ClientConfiguration
4  {
5      public String getConnectionURL();
6  }
7
```

The table below displays the current URLs for CDS.

Server Name	**URL**
Test	https://ap-artdataservicetst-priv.nga.gov:9090
Production Private Data	https://ap-artdataservice-priv.nga.gov:9090

Connection API

Actual connection with CDS is handled by the "DataServiceManager" (gov.nga.common.rpc.DataServiceManager) Singelton class.

```java
import gov.nga.common.rpc.DataServiceManager;
import gov.nga.common.rpc.client.ClientConfiguration;

public class ExhibitionTestingClient
{
    final private DataServiceManager manager;

    public ExhibitionTestingClient(final ClientConfiguration config)
    {
        manager = DataServiceManager.getManager(config);
    }

    public void runTest()
```

The DataServiceManager is comprised of two Service Interfaces

- ArtDataQuerier (gov.nga.common.entities.art)
  This interface is a continuation from version 1 of The Commons Project. These are the principal methods for retrieving TMS Data Objects.

- DataServiceQuerier (gov.nga.common.rpc)
  This interface extends ArtDataQuerier and contains methods meant to replace the role that the local DataManager and Cacher served in version 1 of The Commons Project.

```java
public interface DataServiceQuerier extends ArtDataQuerier
{

    public Date getLastSyncTime();

    public Collection<Long> getAllArtObjectIDs();
    public Collection<Long> getAllExhibitionIDs();
    public Collection<Long> getAllConstituentIDs();
}
```

We will focus a little more now on the methods in DataServiceQuerier.

- **getLastSyncTime()**
  In the future, CDS should be able to real time communicate with subscribers about events like when the data cache updates. But for now, the responsibility is on the Java Client if it has routines dependent on refreshing when the cache does. This method will return a Date object with the timestamp of the last CDS cache sync. CDS generally syncs once a day, but it is probably good practice for Java projects to check this method a few times a day in case the cache updates due to an off schedule task.

- **getAll….IDs()**
  For Java Projects that need to do their own data collection work, these methods take the place of the ability to call the ArtDataCacher directly in version 1 of The Commons Project. These methods only return ids which allow each Java Project to manage data object retrieval from both a memory and network performance perspective. There is a corresponding fetchBy…IDs method in the ArtDataQuerier for each of these data types. One recommended approach is to

break the id list into segments of 25 – 40 ids at a time to send to the fetch routines (this can even be threaded to increase performance).

Design Recommendations

While great effort is taken to keep the Commons API static, there will still be changes over time. To limit the amount of refactoring needed for individual projects, we strongly recommend using a Façade Pattern for implementing integrations with the Core Data Server.

In the "core" bundles of this project, you will find a demonstration of this.

CoreDataService Interface

`llc.alersconsulting.core.client.CoreDataService is used to define the AEM Service for connection to the CDS. In this example, the interface extends DataServiceQuerier (for the CDS search API) as well as ArtDataManagerNotifier (for allowing other project components to subscribe for cache update events). Other project specific interfaces or methods can be added here for centralization purposes.`

CoreDataServiceFacade

`llc.alersconsulting.core.client.CoreDataService is the implementation of CoreDataService. There are a few things to note here:`

Implemented Interfaces

Along with implementing CoreDataService, this class also implements Runnable (for scheduling how often it should check for CDS cache updates and ClientConfiguration (for handling the Configuration properties of the DataServiceManager).

Activation

The activate method is used to obtain an instance of the DataServiceManager class, establish the queue for subscribers, and schedule the initial checking of the cache for updates.

Run()

This method is purely for checking if CDS has updated its cache. Most of the work is done in the called "cacheHasUpdated()" method. An internal datestamp is kept for tracking the last time this component checked the cache status. That is then compared to the timestamp from the DataServiceManager.getLastSyncTime() method. When the CDS timestamp is later, the process for notifying the subscribers is initiated.

DataServiceQuerier methods

The Façade also handles implementing (project side) the API methods for the querying CDS. While this example just forwards the request the DataServiceManager, the abstraction layer here is still very useful. It allows future business rule changes (such as something that may require the use of one of the object Factories) to be implemented here instead of refactoring every Component that has a query affected by the change. It also centralizes the point of connection between the DataServiceManager api and the AEM's project code.

Cache Subscribers

The “llc.alersconsulting.core.client.testers” package contains a few examples of DataManager Subscribers that use the notification to run test on the updated cache.

ArtObjectTester

The CoreDataSerice reference (because of the Façade centralization) is all that this component needs to know to query CDS.

This component also implements the ArtDataManagerSubscriber interface for receiving notifications when the CDS cache has updated. In the artDataUpdated() method, the test are executed.