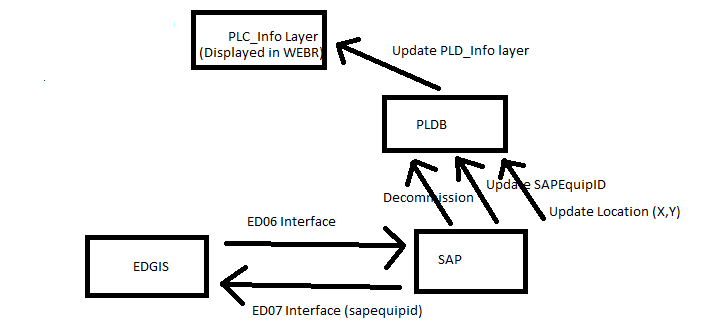
# PLDB Batch Process Design



# Background:

The PLDB batch process provides an alternative to maintaining interfaces between:

1. SAP and the PLDB

2. The PLDB and the PLD\_Info layer (that the estimators see in WEBR)

And will basically replace the arrows in the above diagram between SAP and the PLDB, and between the PLDB and the PLD\_Info layer in WEBR.

## The advantages of this approach are:

* It is not incremental, meaning that it will synchronize the representations of **ALL** poles every time it runs rather than looking at the updates that have been made since the previous day. This means that once the process has run the all updates have been synchronized between the systems and the data should be in complete agreement.
* It does not rely on web services and hence web servers to be online, so there are less points of failure
* The solution will scale really well and will not suffer from problems associated with processing bandwidth (timeouts etc), for example if a mass update were to occur in EDGIS through the running of a script, the batch process will have no problem in completing the required synchronization.
* It will also update elevations in the PLDB where poles have moved, deriving the elevations from the 30m DEM raster layer in the enterprise landbase which represents the heights above sea level for California (if a more fine grained and accurate representation of elevations were required it could be achieved by simply switch out this layer in the Ellevations.mxd map used by the application)
* It will also update the globalids in the PLDB, where the interfaces only updated the sapequipids

# The Basic Design of the Solution

The batch process is a .NET C# executeable, that can be run from the command line. The solution relies on the creation of two major tables within the PLDB. Those tables are:

dbo.SYNC\_EDGIS\_POLES

dbo.SYNC\_PLDB\_POLES

Basic schema of these 2 tables is almost identical and includes:

* PLDBID field
* PGE\_Globalid field
* PGE\_SapEquipId field
* Latitude field
* Longitude field

These two tables represent the EDGIS and PLDB representations of all poles with PLDBIDs within the system (roughly 2.5 million poles). By comparing representations of the x,y locations and elevations of Poles (and PLDBIDs) within these two tables it is possible to determine:

1) Where a Pole has been moved. This would require the location of the PLDBID to be updated in the PLDB as the EDGIS is the source of record for Pole location (here the elevation would also be updated in the PLDB by determining the height above sea level for the new location from the landbase DEM layer)

2) Where a Pole has been deleted without subsequent replacement (where the PLDBID has disappeared from the EDGIS and is still present in the PLDB). This would require that the PLDBID be decommissioned in the PLDB to allow the system to remain synchronized.

3) Where a pole has been switched out or replaced in EDGIS. This scenario is evident by comparing our two tables because the SapEquipId and the edgis Globalid will change but the PLDBID will remain the same, and be inherited by the new Pole. This is consistent with the idea that the PLDBID actually represents the ‘hole in the ground’ at the point where the Pole is installed.

These two tables (sync\_edgis\_poles and sync\_pldb\_poles ) are re-created each time the process is run. The sync\_edgis\_poles is populated using a .NET bulk insert. This design pattern is repeated a number throughout the application.

The basic workflow of the PLDB synchronization is this:

1. PopulateEDGISPoles – populates all edgis poles using the bulk insert (shapes are projected to WGS 84 spatial reference in order to populate the latitude and longitude fields (usually complete within about 12 minutes)

2. PopulatePLDBPoles – populated with a basic SELECT INTO statement which joins the status table and the OCalcProAnalysis table to also include the current status for each PLDBID

3. UpdateLocation – determine which poles have moved by joining the SYNC\_EDGIS\_POLES table and the SYNC\_PLDB\_POLES table on the PLDBID where there is a delta in latitudes or longitudes for a given PLDBID that is more than a very small amount in decimal degrees (0.000001). Those PLDBIDs for which a significant delta in latitude or longitude exists are then loaded into a PLDB table called SYNC\_LAT\_LONG\_DISCREPANCIES. A PLDB stored procedure in the PLDB is then invoked to update the Lat / Long and elevations in the PLDB system tables.

4. Decommission – here a query is done again on the two core tables to find all the PLDBIDs that exist in the SYNC\_PLDB\_POLES but that no longer exist in the SYNC\_EDGIS\_POLES and that are not already reflecting a current status of decommissioned. These PLDBIDs are then decommissioned in the PLDB. This is done by loading the objects identified for decommissioning into an array and making a direct call to the Osmose API.

5. UpdateIDs – this is again very similar to the similar workflows. A join is made between the SYNC\_EDGIS\_POLES and the SYNC\_PLDB\_POLES based on the PLDBID where the sapEquipId or the Globalid does not match. Again the Osmore API is called to bring the keys into sync.

# Synchronization of the PLD\_Info Layer

The synchronization of the PLD\_Info layer would be designed to run in the overnight batch window after the above process that performs the move location, decommission and updating of IDs. The above process basically puts the PLDB into sync with EDGIS, and after this is achieve the synchronization of the PLD\_Info can be run which essentially takes the up-to-date PLDB and generates the entire PLD\_Info layer (approximately 2.5 million poles) using the PLDB as the input. This will negate the need for any messaging or flow of updates between the PLDB and the PLD\_Info WEBR layer.

The design of this process is similar to the design of the previous processes. First a .NET Bulk Insert is used to bring all of the data from the PLDB for all poles into a temporary table: PLD\_INFO\_TEMP. It also populates two extra field the X\_VALUE and the Y\_VALUE in this temp table. These fields are used to derive the shape using the ST\_Point convention. A temporary table is necessary because the OBJECTID value needs to be derived by a sequence and cannot be ported across in a .NET Bulk Insert.

Once the PLD\_INFO\_TEMP table is created the PLD\_INFO layer can then be derived through a SELECT INTO statement. There are two important configuration settings that will govern the successful generation of the PLD\_INFO featureclass. These settings can be seen in the App.config file:

<add key="PLD\_INFO\_OBJECTID\_SEQUENCE" value="R3402812"/>

<add key="PLD\_INFO\_SRID" value="2"/>

The PLD\_INFO\_OBJECTID\_SEQUENCE value tells the select into statement which sequence should be used to generate the OBJECTID in the PLD\_INFO featureclass.

The PLD\_INFO\_SRID is the spatial reference id for the featureclass representing the NAD 83 UTM Zone 10 spatial reference (the value is 2). If the wrong SRID is used the points will not line up with the corresponding supportstructures. The SRID is an important parameter in the constructor for the ST\_Point shape field.

sde.st\_point('point ('|| X\_VALUE ||' '|| Y\_VALUE ||')', 2)