BPM Event and Semantic Data Model Developer’s Guide

Randall Kolb

April 10, 2014

# Table of Contents

[Table of Contents 2](#_Toc384889008)

[1 Overview 4](#_Toc384889009)

[1.1 Workbook 4](#_Toc384889010)

[1.2 Data Models 5](#_Toc384889011)

[1.2.1 BPM Event 5](#_Toc384889012)

[1.2.2 BPM Semantic 7](#_Toc384889013)

[1.3 Queue Processing 8](#_Toc384889014)

[1.3.1 Queue 8](#_Toc384889015)

[1.3.2 Queue Archive 8](#_Toc384889016)

[1.3.3 Processor 9](#_Toc384889017)

[1.3.4 Processor Controller 9](#_Toc384889018)

[2 Development 10](#_Toc384889019)

[2.1 Create Files for Each New Project 11](#_Toc384889020)

[2.1.1 Populate BPM\_PROJECT\_LKUP Table 11](#_Toc384889021)

[2.1.2 Populate BPM\_D\_OPS\_GROUP\_TASK Table 12](#_Toc384889022)

[2.1.3 Create Project-Specific Views for MicroStrategy MASH Reporting 13](#_Toc384889023)

[2.2 Populate Attribute and Lookup Tables 14](#_Toc384889024)

[2.2.1 Populate Process and Source Lookup Tables 15](#_Toc384889025)

[2.3 Triggers 17](#_Toc384889026)

[2.4 BPM Semantic Data Model 20](#_Toc384889027)

[2.4.1 Logical Dimensional Model Design Considerations 20](#_Toc384889028)

[2.4.2 Diagram 23](#_Toc384889029)

[2.4.3 DDL 25](#_Toc384889030)

[2.5 Process Package 26](#_Toc384889031)

[2.6 Truncate 34](#_Toc384889032)

[3 Test 35](#_Toc384889033)

[4 Install 36](#_Toc384889034)

[4.1 Oracle Software 36](#_Toc384889035)

[4.2 Oracle Database 38](#_Toc384889036)

[4.3 Create Connections to System and Developer Users. 40](#_Toc384889037)

[4.4 Maxdat Tablespaces and Users 41](#_Toc384889038)

[4.5 Core ETL and BPM Data Objects 42](#_Toc384889039)

[4.6 Per Project Deployed 42](#_Toc384889040)

[4.7 Per Process Deployed 42](#_Toc384889041)

[5 Administration 43](#_Toc384889042)

[5.1 MAXDAT\_ADMIN Package 43](#_Toc384889043)

[5.1.1 Recompile a program unit natively. 44](#_Toc384889044)

[5.1.2 Configure a BPM calculation job. Enable or disable a job. 44](#_Toc384889045)

[5.1.3 Configure ETL processing. 44](#_Toc384889046)

[5.1.4 Configure the BPM job controller. 45](#_Toc384889047)

[5.1.5 Configure BPM queue processing. 45](#_Toc384889048)

[5.1.6 Display DBMS Scheduler job names. 46](#_Toc384889049)

[5.1.7 Fix metadata of incorrect BPM queue jobs. 46](#_Toc384889050)

[5.1.8 Gather table statistics. 46](#_Toc384889051)

[5.1.9 Reset all BPM processing queue rows that were reserved but failed to process. 47](#_Toc384889052)

[5.1.10 Reset BPM processing queue rows that were reserved but failed to process by BPM Source Lookup ID. 47](#_Toc384889053)

[5.1.11 Reset BPM processing queue rows that were reserved but failed to process by BPM Queue PROCESS\_BUEQ ID. 47](#_Toc384889054)

[5.1.12 Shutdown all BPM control, queue and calculation jobs. 48](#_Toc384889055)

[5.1.13 Startup all BPM control, queue and calculation jobs. 48](#_Toc384889056)

[5.1.14 Stop a DBMS Scheduler job. 48](#_Toc384889057)

[5.2 Local Personal Oracle Database on a Windows PC 49](#_Toc384889058)

[5.2.1 Oracle Listener 49](#_Toc384889059)

[5.2.2 Oracle Database 50](#_Toc384889060)

[5.3 Manually Stop Jobs 51](#_Toc384889061)

[5.3.1 List running DBMS Scheduler jobs. 51](#_Toc384889062)

[5.3.2 Stop a running DBMS Scheduler job. 51](#_Toc384889063)

[6 Glossary 52](#_Toc384889064)

# Overview

This is a developer’s guide for writing code to populate the Maximus Business Process Management BPM Event and Semantic data models. It consists of an overview of how things work, a development section describing the implementation steps and how to test and deploy.

## Workbook

Each process that is monitored has a requirements workbook that is an Excel spreadsheet that is available via a project repository.

* Example: <https://maxcs.maxinc.com/gm/folder-1.11.616772>

The workbooks contain info on the processes, process flow charts, ETL requirements, staging tables, attributes and report requirements. These workbooks are created and maintained by a project analyst who is the go-to person for any questions you have about requirements for this process.

## Data Models

Data from the staging table or other data source such as event-driven is eventually stored in the BPM Event and Semantic data models.

### BPM Event

The BPM Event data model is a normalized data model containing a record of current and much historical info on the attributes values of the processes monitored. This is the database of record for BPM. MicroStrategy reports do not use these tables as performance is slow due to multiple table joins required for reports using this model.

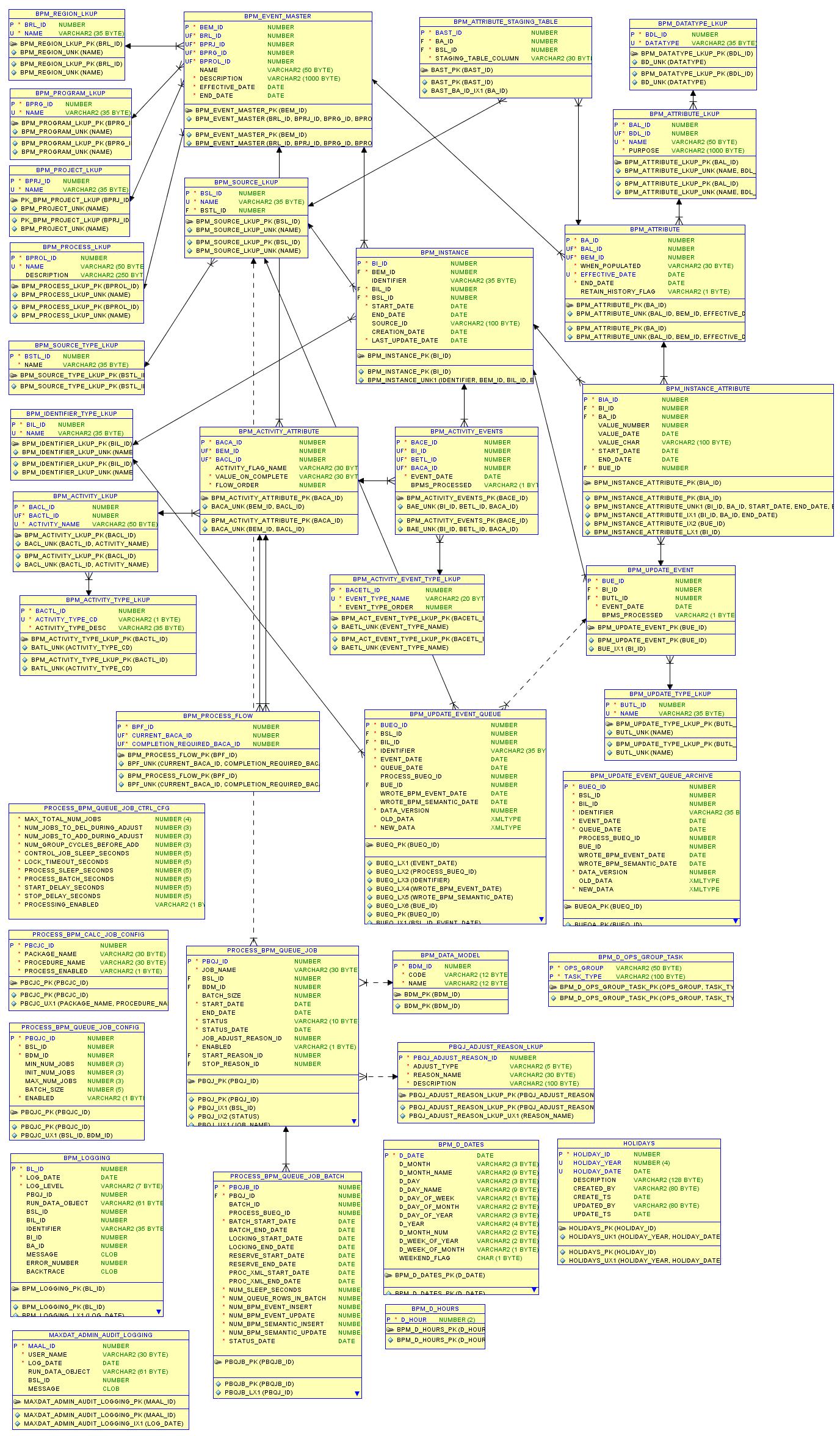
The BPM Event data model diagram can be found at: (and also on the next page of this document)

BPM/DataModel/BpmEvent/doc/BPM\_Event\_data\_model\_diagram.pdf

The data model design source for a diagram can be found by unzipping

BPM/DataModel/BpmEvent/doc/BPM\_Event\_data\_model\_diagram.zip

The BPM Event data model diagram is shown on the next page of this document.



### BPM Semantic

The BPM Semantic data model is a star schema dimensional model that contains a record of current and some historical info on the attributes values of the processes monitored. MicroStrategy reports use views of these tables.

Each project has its own semantic data model diagram at

BPM/{project}/doc/{project}\_BPM\_Semantic\_data\_model\_diagram.pdf

Example:

BPM/ILEB/doc/ILEB\_BPM\_Semantic\_data\_model\_diagram.pdf

The data model design source for a diagram can be found by unzipping

BPM/{project}/doc/{project}\_BPM\_Semantic\_data\_model\_diagram.zip

Example:

BPM/ILEB/doc/ILEB\_BPM\_Semantic\_data\_model\_diagram.zip

## Queue Processing

Data from staging tables and event driven is written as XMLs to the BPM Queue table. It is then processed and written to the BPM Event and Semantic data models.

TBA

### Queue

The BPM queue table is BPM\_UPDATE\_EVENT\_QUEUE. It stores the XML representation of selected staging table columns with before and after insert/update states. This table is populated by after insert/update triggers on the final staging tables for each process. This queue is processed later by queue processing jobs to populate the BPM Event and Semantic data models.

The BPM\_UPDATE\_EVENT\_QUEUE table is defined in svn://rcmxapp1d.maximus.com/maxdat/BPM/EventQueue/createdb/BPM\_UPDATE\_EVENT\_QUEUE.sql

### Queue Archive

After a queue row is processed it is written to the BPM queue archive table BPM\_UPDATE\_EVENT\_QUEUE\_ARCHIVE and removed from the queue table.

Records with an EVENT\_DATE > 7 days can be removed. No automated process does this currently. Usually they are removed by not being part of the select into a temp table that is part of cleaning out an entire BSL\_ID in a truncate\_data\_model\_\*.sql script run.

The BPM\_UPDATE\_EVENT\_QUEUE\_ARCHIVE table is defined in svn://rcmxapp1d.maximus.com/maxdat/BPM/EventQueue/createdb/BPM\_UPDATE\_EVENT\_QUEUE.sql

### Processor

Processor jobs run nearly continuously that select, reserve and process BPM queue rows. Data from a queue row is first written to the BPM Event data model, and then if successful, written to the BPM Semantic data model. Process jobs are run as multiple DBMS\_SCHEDULER jobs in infinite loops that look for and process batches of available queue rows.

TBA

Configuration

TBA

Metrics

TBA

### Processor Controller

The lifespan, number and type of data processed of processors running is controlled by the processor controller.

It continually adjusts the number of processors based on configuration, reservation lock contention, and number of queue rows of a type remaining to be processed.

TBA

# Development

To develop code to populate the BPM Event and Semantic data models, read discussion below, copy examples below as templates and follow implementation steps. Development work should be done in order presented below.

The examples below use the NYEC project and the ProcessApp process. Processes that are shared by multiple projects are considered to be part of the Corp project. In that case, use Corp as your project and create a folder in BPM/Corp for your process files. Then use BPM/Corp/ManageWork as your example process instead.

**When looking at other projects and processes for examples always get the example data objects from the SVN files that were run to create these. Do not extract from the database. You may lose important configuration and compile steps.**

**It is important that the code match the naming standards, format and logical flow of the examples.** This makes it easier to review code and debug problems and to later make improvements across all processes. Match the examples closely the first time you code a process. If you have ideas for improvements, share them with your manager so that it can be implemented for all processes. Sometime in the future the code may be more automated, consistent and standardized code will make this easier.

A canonical database, MAXDTDEV, is used as the central repository of all lookup IDs used by all projects (such as a list of all project IDs (BPRJ\_ID) or attribute IDs (BA\_ID)). Individual project databases store only the codes used by the project. The canonical database is used as the one place where the next available (unused) IDs can be calculated so that an ID avoids having a different meaning on different databases.

## Create Files for Each New Project

Only required when creating a new project.

Example: ILEB, NYEC, TXEB, etc.

### Populate BPM\_PROJECT\_LKUP Table

Example: BPM/NYEC/createdb/populate\_BPM\_PROJECT\_LKUP.sql

Implementation Steps:

Copy example and use as a template to create project file:

BPM*/{project}* /createdb/populate\_BPM\_PROJECT\_LKUP.sql

Edit insert statement:

BPRJ\_ID = next available BPRJ\_ID

Get next available BPRJ\_ID:

* + - * (run on the canonical db)
      * select max(BPRJ\_ID) from BPM\_PROJECT\_LKUP;

NAME = get from analyst

### Populate BPM\_D\_OPS\_GROUP\_TASK Table

Example: BPM/NYEC/createbd/populate\_BPM\_D\_OPS\_GROUP\_TASK.sql

Implementation Steps:

* Copy example and use as a template to create project file:

BPM*/{project}* /createdb/populate\_BPM\_D\_OPS\_GROUP\_TASK.sql

* Get spreadsheet “\* Task Type Groups” from project’s workbook repository.
* Create insert statements by adding a new formula column named “SQL” (if not already there) to workbook in row 2:

=CONCATENATE("insert into BPM\_D\_OPS\_GROUP\_TASK (OPS\_GROUP,TASK\_TYPE) values ('",$G2,"','",$A2,"');")

Example: <https://maxcs.maxinc.com/gm/document-1.9.1092837/NYEC_Task_Type_Groups.xlsx>

* Replace $G in formula with column letter for OPS\_GROUP value.
* Replace $A in formula with column letter for TASK\_TYPE value
* Copy formula to every data row.
* Excel will calculate and create an insert SQL statement for each row.
* Copy insert statements and replace existing inserts in populate\_BPM\_D\_OPS\_GROUP\_TASK.sql

### Create Project-Specific Views for MicroStrategy MASH Reporting

Example: BPM/NYEC/createbd/MASH\_views.sql

Implementation Steps:

* Copy example and use as a template to create project file:

BPM*/{project}* /createdb/MASH\_views.sql

* Replace example project name with new project name for all 5 views.

Example:

Change:

*create or replace view D\_****NYEC****\_BPM\_SOURCE\_LKUP\_SV as*

*select \* from D\_BPM\_SOURCE\_LKUP\_SV*

*with read only;*

To:

*create or replace view D\_****ILEB****\_BPM\_SOURCE\_LKUP\_SV as*

*select \* from D\_BPM\_SOURCE\_LKUP\_SV*

*with read only;*

## Populate Attribute and Lookup Tables

Populate lookup tables used by projects and processes. Currently MAXDTDEV is the canonical database for keeping track of lookup ID usage so that we avoid duplicating lookup values. Make sure all of these values are deployed there in addition to program-specific dev, test, UAT and production databases. A database (with exception of canonical MAXDAT) only has lookup values for processes related to its project.

Example:

BPM\_PROCESS\_LKUP.BPROL\_ID = 2 refers to “NYEC Process App”,

This value will be present in databases MAXDTDEV and NYECMXD\*

but not ILEBMXD\* or FLHKMXD\*.

When a new process is implemented and deployed, its BPM\_PROCESS\_LKUP.BPROL\_ID value will be the next available ID number:

(run on MAXDTDEV db)

select max(BPROL\_ID ) + 1 from BPM\_PROCESS\_LKUP;

### Populate Process and Source Lookup Tables

Populate tables used to lookup general info about the process and staging table.

Example: BPM/NYEC/ProcessApp/createdb/populate\_lkup\_tables.sql

Implementation Steps:

Copy example and use as a template to create file:

BPM*/{project}* */{process}* /createdb/populate\_lkup\_tables.sql

Create insert SQL for BPM\_PROCESS\_LKUP to describe the process.

* + Get BPROL\_ID as next available ID:
    - (run on the canonical db)
    - select max(BPROL\_ID) + 1 from BPM\_PROCESS\_LKUP;
  + Set NAME as appropriate.

Create insert SQL for BPM\_SOURCE\_LKUP to describe the staging table or other data source such as event-driven source.

* + Get BSL\_ID as next available ID:
    - (run on the canonical db)
    - select max(BSL\_ID) + 1 from BPM\_SOURCE\_LKUP;
  + NAME is the staging table name in the case of a staging table source.

Create insert SQL for PROCESS\_BPM\_QUEUE\_JOB\_CONFIG to configure the jobs for queue processing. Create one entry for each data model (BDM\_ID).

BDM\_ID = 2 for BPM Semantic

Create insert SQL for PROCESS\_BPM\_CALC\_JOB\_CONFIG to configure the jobs for semantic calculation processing. Usually there is only one entry for each process and only an entry if there is a calculation procedure in the process. Creating this calculation procedure is done in Section 2.4 Process Package.

Example:

insert into PROCESS\_BPM\_CALC\_JOB\_CONFIG (PBCJC\_ID,PACKAGE\_NAME,PROCEDURE\_NAME,PROCESS\_ENABLED)

values (SEQ\_PBCJC\_ID.nextval,'NYEC\_PROCESS\_APP','CALC\_DNPACUR','Y');

Create insert SQL for BPM\_EVENT\_MASTER to describe the process

and map the project to the process.

If this is a shared Corp process (example Manage Work) then, instead, create a new file to hold this insert called:

{project}/{process}/created/populate\_BPM\_EVENT\_MASTER.sql

Example:

ILEB/ManageWork/created/populate\_BPM\_EVENT\_MASTER.sql

* + Get BEM\_ID as next available ID:
    - (run on the canonical db)
    - select max(BEM\_ID) + 1 from BPM\_EVENT\_MASTER;
  + Get BRL\_ID as appropriate region value for this process.
    - (run on the canonical or one of this project’s dbs)
    - select \* from BPM\_REGION\_LKUP;
  + Get BPRJ\_ID as the appropriate project value for this process.
    - (run on the canonical db or one of this project’s dbs)
    - select \* from BPM\_PROJECT\_LKUP;
  + Get BPRG\_ID as the appropriate program value for this process.
    - (run on the canonical db or one of this project’s dbs)
    - select \* from BPM\_PROGRAM\_LKUP;
  + Get BPROL\_ID as the appropriate process value for this process.
    - Use value you created at the start of working on this populate\_lkup\_tables.sql script.
  + Set NAME and DESCRIPTION as appropriate.

## Triggers

Three triggers are created for each final ETL staging table (example: CORP\_ETL\_MANAGE\_WORK).

* a before insert or update to set IDs and dates by default
  + **TRG\_BIU\_*{****staging\_table}*
* an after insert to write a snapshot of the changes to the queue table
  + **TRG\_AI\_*{****staging\_table}***\_Q**
* an after update to write a snapshot of the changes to the queue table
  + **TRG\_AU\_*{****staging\_table}***\_Q**

The two triggers write snapshot by writing an XML of the old and an XML of the new row data, plus metadata to the BPM\_UPDATE\_EVENT\_QUEUE queue table. The queue table is the common source used to populate the BPM Event and Semantic data models.

Implementation Steps:

Copy example and use as a template to create file:

BPM*/{project}* */{process}* /createdb/create\_ETL\_{process}\_triggers.sql

Example:

BPM*/Corp/ManageWork/*createdb/create\_ETL\_ManageWork\_triggers.sql

Standardize indents and line spacing so that the code is readable.

Before trigger (**TRG\_BIU\_*{****staging\_table}* )

* + Replace unique staging table system-generated source ID (Example: CEMW\_ID) with similar ID for new staging table.
  + Set other fields such as appropriate for new staging table.

Queue triggers (**TRG\_A***{type}***\_*{****staging\_table}***\_Q) :**

Edit declare block:

v\_bsl\_id – get BPM\_SOURCE\_LKUP.BSL\_ID value from populate\_lkup\_tables.sql

v\_bil\_id – get BPM\_IDENTIFIER\_TYPE\_LKUP.BIL\_ID value

from populate\_lkup\_tables.sql or db if reused from other process

v\_data\_version – is 1 at start for each process, increment when XML stored in

BPM\_UPDATE\_EVENT\_QUEUE changes format, such as adds or removes XML

elements (staging table columns), keep comments for each data version

Set Identifier

* Determine unique identifier column(s) for this trigger’s staging table.
  + Look in requirements workbook in Scope Statement tab,:
    - Examine value for “What is the instance?” row and compare to staging table column via Instance Attributes and Staging Table tabs.
* Set IDENTIFIER value to appropriate unique identifier for the trigger’s staging table.

Examples:

* + - * + TASK\_ID
        + APP\_ID || '\_' || REACTIVATION\_NBR;
* Edit commented-out XML string SQL block(s):

This SQL is later run manually on the database with the staging table to create the XML element string for the trigger.

The insert trigger references new: data from the staging table.

The update trigger has 2 SQL blocks, one for the :old data and one for the :new data (before and after states of the staging table).

Dates are stored in the standardized format defined in BPM/Core/created/BPM\_COMMON\_pkg.sql value DATE\_FMT (defined as ‘YYYY-MM-DD HH24:MI:SS’).

All varchar2 data must use CDATA wrappers to avoid issues with XML special characters such as ‘<’.

* Add any non-attribute required fields such as source ID and the event date
  + These are fields that are not stored in BPM\_ATTRIBUTE but are used by BPM Event and Semantic data models
    - Example: CEMW\_ID and STG\_LAST\_UPDATE\_DATE
* Edit where clause to select attributes for this process:
  + Example: bast.BSL\_ID = 1 and atc.TABLE\_NAME = ‘CORP\_ETL\_MANAGE\_WORK’

Run XML String SQL block(s) on the database with the staging table and paste results into trigger code:

* Copy SQL block code and execute SQL.
* Paste results:
  + <ROW>
  + *{paste results}*
  + </ROW>

This XML string has a flat structure that represents row of the staging or source data and uses the staging table column names as the XML element names.

## BPM Semantic Data Model

A star-schema dimensional model is created for each process.

### Logical Dimensional Model Design Considerations

Define the shorthand process name and instance identifier.

Example: for “NYEC Process Application”, the shorthand name is “NYEC\_PA”

Each process instance is identified by an instance ID, BI\_ID.

The BI\_ID in the dimensional model should be named *{process\_shorthand\_name}*\_BI\_ID.

Example: NYEC\_PA\_BI\_ID

Create a “Current” dimension table named for the process.

This should be named D\_*{process\_short\_name\_name}*\_CURRENT

The table name should be <= 27 characters. This is so that a “\_SV” can be appended to the table name when a view of it is created later.

Example: D\_NYEC\_PA\_CURRENT

Create a column for the BI\_ID (see above):

Example: NYEC\_PA\_BI\_ID

The current dimension table should contain the process instance identifier and all attributes defined in the requirements workbook in the Populate Attribute Lookup table above. Also include rows that were not used earlier because the When Populated column had the value “CALCULATION”. They are used here.

For each attribute, create a column:

* + Name columns to match attribute name or abbreviated in a clear manner so that the column name is 30 characters or less.
    - Example attribute: “Incoming Scanned Document Processing Owner Name”
      * Good column name: IN\_SCANNED\_DOC\_PROC\_OWN
      * Bad column name: “INCM\_ SCNNED\_DOCMNT\_PRO\_OWNR\_NM”
        + (misspelling-prone abbreviations)
  + Note that older processes used double quoted , mixed case column names (example: “In Scanned Doc Proc Own” ). That format not used for new processes or additions to older process.
  + Column types, lengths and not null constraints should match the source staging table.
  + For attributes that store historical data

(BPM\_ATTRIBUTE.RETAIN\_HISTORY\_FLAG = ‘Y’):

* + - Prepend “Cur “ to the column name, using rules above:

Example: “ IN\_SCANNED\_DOC\_PROC\_OWN

* + - For attributes that are calculated, no historical data is stored.

Create separate dimensions for attributes that store historical data.

* + Except date/times which should be stored in the fact table.
  + Group attributes together in a dimension that have a common relationship.
    - Example:
      * Dimension D\_NYEC\_PA\_APP\_STATUS has: (old double-quoted naming format)
        + "App Status"
        + "App Status Group”
        + "Heart App Status"
        + "Refer to LDSS Flag"
  + Add a unique sequence-generated ID column to each historical data dimension.

Using the format D*{initial\_of\_project}{initial\_of\_process}{initials\_of\_attribute\_group}*\_ID

* + - Example: DNPAAS\_ID

Create a fact table.

* + Determine data consolidation level for the fact table (default: by day).
  + Contains :
    - a sequence-generated ID, using the formula F*{initial\_of\_project}{initial\_of\_process}{initials\_of\_attribute\_group}*\_ID
      * Example: FNPAAS\_ID
    - date/time of the fact event
      * D\_DATE
    - range of date/time that this fact represents
      * BUCKET\_START\_DATE
      * BUCKET\_END\_DATE
    - a column for the BI\_ID :
      * Example: NYEC\_PA\_BI\_ID
    - IDs for each historical data dimension
    - date/time fields for all date/time historical attributes
    - metrics
      * CREATION\_COUNT
      * INVENTORY\_COUNT
      * COMPLETION\_COUNT

Determine attributes that represent identifiers which are common to other processes.

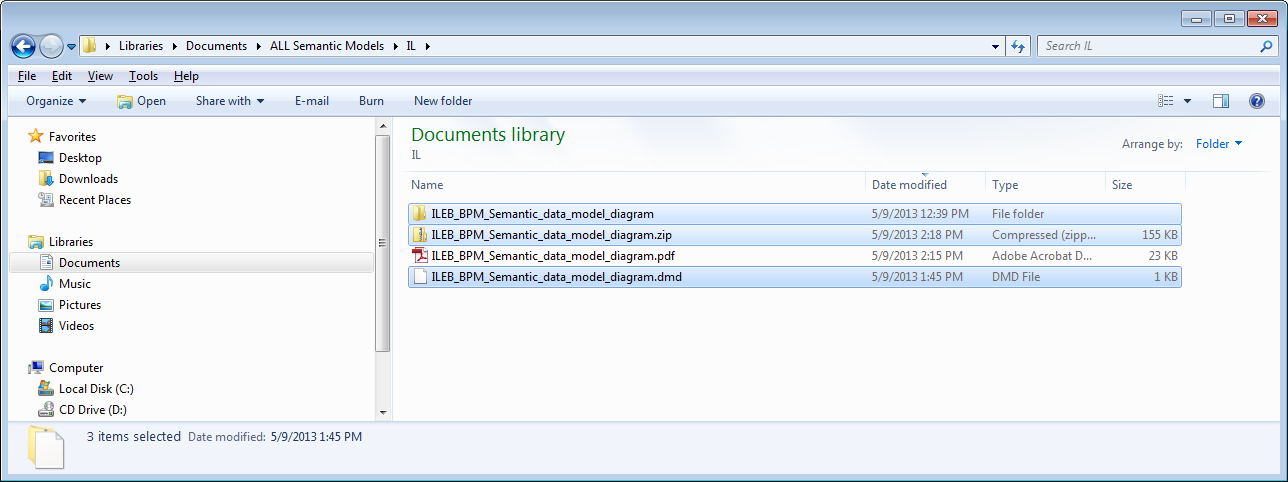
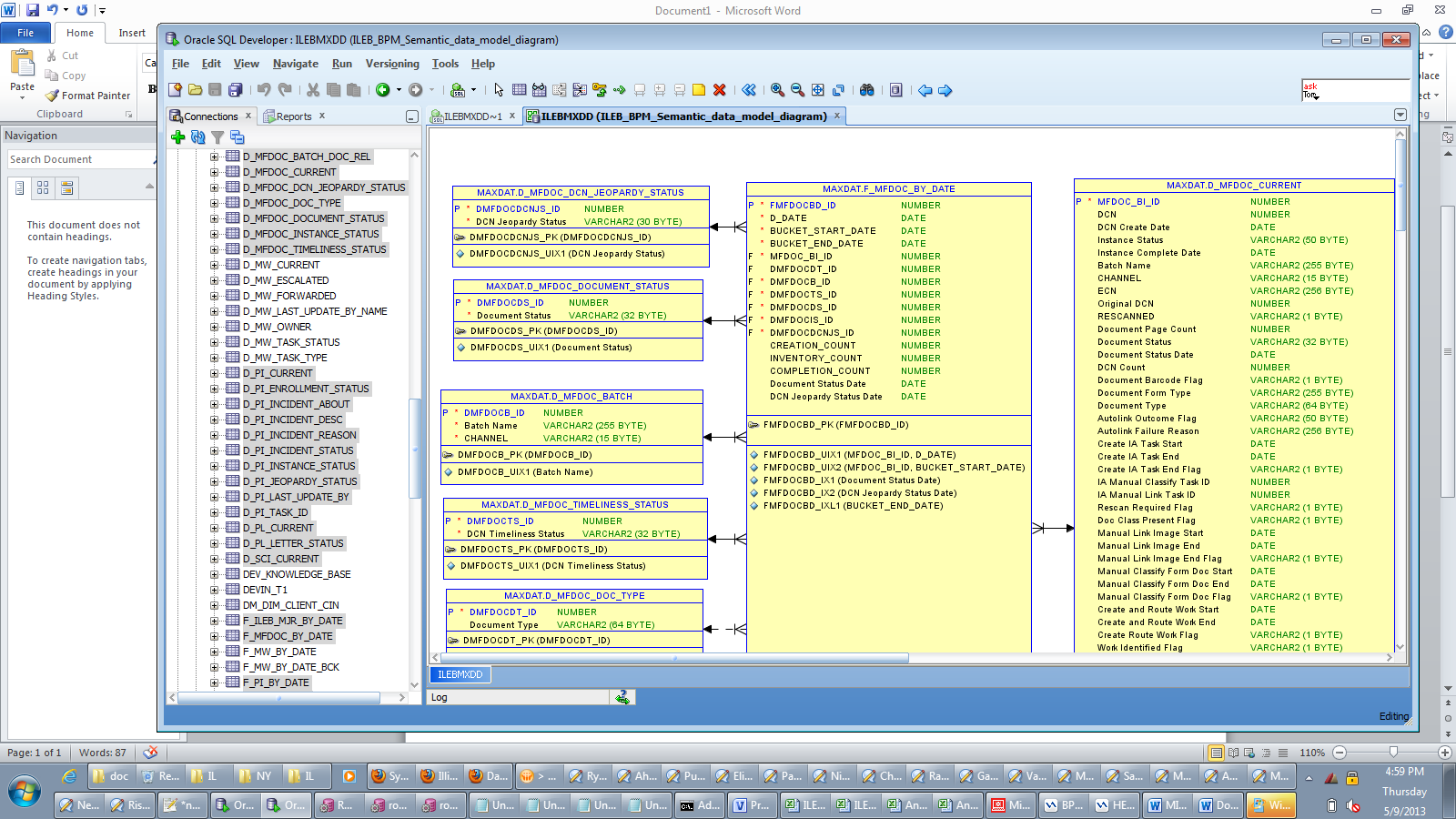
* + Example:
    - NYEC Process App contains an attribute “Current Task ID” which could be linked to “Task ID” in the Manage Work process.
  + For each relationship to another process where details from the other process are desired, create a dimension containing all attributes of the “Current” dimension from the related process
    - Example:
      * Create a dimension for Process App named “D\_PROCESS\_APP\_CURRENT\_TASK” with an identifier of “Current Task ID” and fields for all attributes in the D\_MANAGE\_WORK\_CURRENT dimension.

### Diagram

Generate a data model diagram from the logical design.

Implementation Steps:

**Loading Model onto Workstation**

* Open SmartSVN (or other visual SVN browser)
* Navigate to a directory with zip file of a database design model .
  + Example: BPM/ILEB/doc
    - (which has ILEB\_BPM\_Semantic\_data\_model\_diagram.zip)
* Right-click on the doc folder and select Update to make sure you have latest version of files.
* Double-click on the zip file.
  + Example: ILEB\_BPM\_Semantic\_data\_model\_diagram.zip
* Select Extract All Files to extract the zip file into a local working folder. Once extracted, you will see a .dmd file and a supporting folder.
  + 
* Open Oracle SQL Developer
* To open the model, select File > Data Modeler > Open -> select .dmd file -> OK
* Select Relational Model
  + Example: ILEBMXDD
* The model will look like this: 
* From here, you can delete objects by right-clicking them and selecting delete or add objects by dragging the table(s)/view(s) onto the model diagram.
* To save the model go to File > Data Modeler > Save As
* To create a PDF version of the model go to File > Data Modeler > Print Diagram > To PDF File

Save diagram as BPM/*{project}*/doc/*{project}*\_BPM\_Semantic\_data\_model\_diagram.pdf

* + - Example: BPM/ILEB /doc/ILEB\_BPM\_Semantic\_data\_model\_diagram.pdf
* Once the model has been updated, be sure and re-zip the saved .dmd file AND the supporting directory structure.
  + - Example: Zip these files and directories as ILEB\_BPM\_Semantic\_data\_model\_diagram.zip
      * ILEB\_BPM\_Semantic\_data\_model\_diagram (directory of files)
      * ILEB\_BPM\_Semantic\_data\_model\_diagram.dmd
* Then save changed zip file and PDF diagram in SVN.

### DDL

Create script for the BPM Semantic data model DDL for this process.

Implementation Steps:

Export the DDL from the data model as file:

BPM*/{project}* */{process}* /createdb/SemanticModel\_*{project}*\_*{process}*.sql

Look at an example BPM Semantic data model DDL script :

BPM/NYEC/ProcessApp/createdb/SemanticModel\_NYEC\_Process\_App.sql

Edit the exported data model DDL so that it matches the format and features of the example script.

Make sure that:

* sequences are created
* tablespace is MAXDAT\_DATA and tables with more than a few hundred rows expected should be created to allow parallel query
* primary key is created and rebuilt on tablespace MAXDAT\_INDX
* create indexes
* create constraints
* create a read only view of the dimensions and named as *{table\_name}*\_SV
* seed dimensions that allow null values to prevent initial deploy race condition as multiple parallel jobs try and populate dimension with null when source has a lot of nulls for that attribute
  + Example:
    - insert into D\_NYEC\_PA\_HEART\_INC\_APP\_IND (DNPAHIAI\_ID,"HEART Incomplete App Ind") values (SEQ\_DNPAHIAI\_ID.nextval,null); commit;
* revise the definition of the exported fact table DDL to match example, which includes partitions
  + Example: F\_NYEC\_PA\_BY\_DATE
* create a view of the fact table to match example:
  + Example: F\_NYEC\_PA\_BY\_DATE\_SV
* formatted neatly and consistently

## Process Package

Create package to process BPM queue rows for this process and load data into the BPM Event and Semantic data models.

Example: BPM/NYEC/ProcessApp/createdb/NYEC\_PROCESS\_APP\_pkg.sql

Implementation Steps:

Copy example and use as a template to create file:

BPM*/{project}* */{process}* /createdb/*{project}*\_*{process}\_*pkg.sql

List functions and procedures in alphabetical order for neatness, unless a definition is needed earlier so that it compiles.

Standardize indents and line spacing so that the code is readable.

Creating this package from the template requires extensive replacement of variable names and values to be relevant to the process. The same structure and order remain for all processes.

* Create package specification.
  + Add SVN keyword to help identify which versions of the code are deployed to databases.
    - Cut and paste these immediately after the package specification header:

**-- Do not edit these four SVN\_\* variable values. They are populated when you commit code to SVN and used later to identify deployed code.**

**SVN\_FILE\_URL varchar2(200) := '$URL$';**

**SVN\_REVISION varchar2(20) := '$Revision$';**

**SVN\_REVISION\_DATE varchar2(60) := '$Date$';**

**SVN\_REVISION\_AUTHOR varchar2(20) := '$Author$';**

* + - Example:

*create or replace package NYEC\_PROCESS\_APP as*

**-- Do not edit these four SVN\_\* variable values. They are populated when you commit code to SVN and used later to identify deployed code.**

**SVN\_FILE\_URL varchar2(200) := '$URL$';**

**SVN\_REVISION varchar2(20) := '$Revision$';**

**SVN\_REVISION\_DATE varchar2(60) := '$Date$';**

**SVN\_REVISION\_AUTHOR varchar2(20) := '$Author$';**

…

* + - When code is committed to SVN these values are populated.

Example:

*create or replace package NYEC\_PROCESS\_APP as*

*-- Do not edit these four SVN\_\* variable values. They are populated when you commit code to SVN and used later to identify deployed code.*

*SVN\_FILE\_URL varchar2(200) := '$URL****: svn://rcmxapp1d.maximus.com/maxdat/BPM/NYEC/ProcessApp/createdb/NYEC\_PROCESS\_APP\_pkg.sql*** *$';*

*SVN\_REVISION varchar2(20) := '$Revision****: 3559*** *$';*

*SVN\_REVISION\_DATE varchar2(60) := '$Date****: 2013-07-01 16:30:27 -0500 (Mon, 01 Jul 2013)*** *$';*

*SVN\_REVISION\_AUTHOR varchar2(20) := '$Author****: rk50472*** *$';*

*…*

* + Enable keyword substitution in new files:
    - When committing the code to SVN enable the SVN keyword substitution.

This step is done later after you first commit this file to SVN.

In SmartSVN:

right-click new file -> Properties -> Keyword Substitution

check Set for:

Author

Date

Revision

URL

OK

Commit changed settings to SVN.

In TortoiseSVN:

Should be similar to SmartSVN.

Please add any differences to this document.

There are several functions and procedures that are in this example but not in other process packages so that some attributes can be calculated rather than received from a data source such as a staging table. You do not need to include procedures and functions like these in the package unless there are calculated attributes required for this specific process.

Examples from NYEC Process App:

* + - procedure CALC\_DNPACUR *(example calculation procedure)*
    - function GET\_AGE\_IN\_BUSINESS\_DAYS
    - function GET\_AGE\_IN\_CALENDAR\_DAYS
    - function GET\_APP\_CYCLE\_BUS\_DAYS
    - function GET\_APP\_CYCLE\_CAL\_DAYS
    - function GET\_JEOPARDY\_FLAG
    - function GET\_TIMELINESS\_STATUS
    - function GET\_DAYS\_TILL\_TIMEOUT

Create data record types that match attributes stored in BPM queue old and new XML data.

Examples: T\_INS\_PA\_XML and T\_UPD\_PA\_XML

* + - Edit the commented out SQL, execute on the database with the staging table and copy and paste results into the package code to get the attributes for the column definition lines. This is similar to what was done in the insert and update triggers on the staging table.
    - Date fields are stored as varchar2(19) to store in the format ‘YYYY-MM-DD HH24:MI:SS’.

Create package body.

* + Set process variables and comments as defined in Populate Process and Source Lookup Tables above.
    - v\_bem\_id
    - v\_bil\_id
    - v\_bsl\_id
  + Rarely you may need to change these values:
    - v\_butl\_id - if the data source is not a staging table, such as event driven
    - v\_date\_bucket\_fmt – if different data consolidation level for the BPM Semantic fact table (usually by day)
  + The procedures and facts listed above in the package header instruction steps above are only needed if some attribute are calculated.
  + Create procedures to get/populate BPM Semantic dimensions.
    - Return dimension unique ID for matching data.
    - Create new dimension row if data doesn’t exists yet.
    - Handle collisions from parallel jobs creating new rows with the same data.
    - Example: GET\_DNPACOU\_ID
  + Create procedure to get data records from the queue XML.
    - Examples: GET\_INS\_PA\_XML and GET\_UPD\_PA\_XML
    - Edit the commented out SQL, execute on the database with the staging table and copy and paste results into the package code to get the attributes for the extractValue lines. This is similar to what was done in the insert and update triggers on the staging table.
  + Modify any data\_version code.

Data version allows different generations of XML to co-exist in the queue or queue archive or during processing. The first release for a process should be data\_version = 1.

The example file has this as “if p\_data\_version > 3 then” in four places. Change this to as “if p\_data\_version > 1 then”.

* + Create procedure to insert data to BPM Event data model.
    - Example: INSERT\_BPM\_EVENT
    - Set variables:
      * v\_identifer = unique identifier for the instance
      * v\_start\_date = create date equivalent
      * v\_end\_date = complete date equivalent
    - Set SOURCE\_ID in insert into BPM\_INSTANCE statement to system generated staging table row ID.
      * Example: CEPA\_ID
    - For each attribute, defined in BPM\_ATTRIBUTES for this process:
      * Add a BPM\_EVENT.INSERT\_BIA statement to insert to the BPM\_INSTANCE\_ATTRIBUTE table.
        + BPM\_ATTRIBUTE.BA\_ID

Example: 37

* + - * + BPM\_ATTRIBUTE\_LKUP.BDL\_ID

Example: 2

* + - * + Data record column

Example: v\_new\_data.APP\_COMPLETE\_RESULT

* + Create procedure to update data to BPM Event.
    - Example: UPDATE\_BPM\_EVENT
    - Set variables:
      * v\_identifer = unique identifier for the instance
      * v\_end\_date = complete date equivalent
    - For each attribute, defined in BPM\_ATTRIBUTES for this process:
      * Add a BPM\_EVENT.UPDATE\_BIA statement to insert and update to the BPM\_INSTANCE\_ATTRIBUTE table.
        + BPM\_ATTRIBUTE.BA\_ID

Example: 37

* + - * + BPM\_ATTRIBUTE\_LKUP.BDL\_ID

Example: 2

* + - * + BPM\_ATTRIBUTE.RETAIN\_HISTORY\_FLAG

Example: ‘Y’

* + - * + Old data record column

Example: v\_old\_data.APP\_COMPLETE\_RESULT

* + - * + New data record column

Example: v\_new\_data.APP\_COMPLETE\_RESULT

* + Create procedure to get dimension data IDs and insert current dimension and fact to BPM Semantic data model.
    - Example: INSERT \_BPM\_SEMANTIC
    - Set variables:
      * v\_identifer = unique identifier for the instance
      * v\_start\_date = create date equivalent
      * v\_end\_date = complete date equivalent
    - Call dimension procedures and get/set dimension IDs from data values.
    - Insert data values into Current dimension.
      * Example: SET\_DNPACUR(…)
    - Insert data values into the fact.
      * Example: INS\_FNPABD(…)
  + Create procedure to insert/update the Current dimension of the BPM Semantic data model..
    - Example: SET\_DNPACUR
    - Create procedure parameters for all attributes
    - Set insert data record values.
      * Example: r\_dnpacur
      * Convert date strings to date, as needed.
      * Perform calculations for any calculated attributes.

(convert dates in varchar2 to Oracle dates where needed)

* + Create procedures to insert fact to BPM Semantic data model.
    - Example: INS\_FNPABD
    - Create procedure parameters for all attributes
    - Set insert data record values.
      * Example: r\_dnpabd
      * Convert date strings to date, as needed.
      * Perform calculations for any calculated attributes.

(convert dates in varchar2 to Oracle dates where needed)

* + - * Edit procedure to match process table name and columns.
  + Create procedure to get dimension data IDs and update fact to BPM Semantic.
    - Example: UPDATE\_BPM\_SEMANTIC
    - Set variables:
      * v\_identifer = unique identifier for the instance
      * v\_start\_date = create date equivalent
      * v\_end\_date = complete date equivalent
    - Call dimension procedures and get/set dimension IDs from data values.

(convert dates in varchar2 to Oracle dates where needed)

* + - Update data values in Current dimension.
      * Example: SET\_DNPACUR(…)
    - Update/insert data values in the fact.
      * Example: UPD\_FNPABD(…)
  + Create procedure to update fact to BPM Semantic.
    - Example: UPD\_FNPABD
    - Create procedure parameters for all attributes
    - Set insert data record values.
      * Example: r\_dnpabd
      * Convert date strings to date, as needed.
      * Perform calculations for any calculated attributes.

(convert dates in varchar2 to Oracle dates where needed)

* + - Edit procedure to match process table name and columns.

Add references in project data model package body.

* Edit files that map project to processing package procedures.
  + BPM/*{project}*/createdb/BPM\_EVENT\_PROJECT\_pkg\_body.sql
    - Example: BPM/NYEC/created/BPM\_EVENT\_PROJECT\_pkg\_body.sql
    - Add entry for process BSL\_ID in INSERT\_BPM\_EVENT if-then-else block.
    - Add entry for process BSL\_ID in UPDATE\_BPM\_EVENT if-then-else block.
  + BPM/*{project}*/createdb/BPM\_SEMANTIC\_PROJECT\_pkg\_body.sql
    - Example: BPM/NYEC/created/BPM\_SEMANTIC\_PROJECT\_pkg\_body.sql
    - Add entry for process BSL\_ID in INSERT\_BPM\_SEMANTIC if-then-else block.
    - Add entry for process BSL\_ID in UPDATE\_BPM\_SEMANTIC if-then-else block.
* Make a copy of these 2 files for deployment purposes.

Otherwise, someone developing another process for this project will add a reference to a process package that you are not yet deploying and this code will fail to compile at deploy time.

* + Copy files:
    - BPM/*{project}*/createdb/BPM\_\* \_PROJECT\_pkg\_body.sql
  + To:
    - BPM/*{project}*/{process}/patch /BPM\_\* \_PROJECT\_pkg\_body.sql

## Truncate

(optional)

Create a script to remove all BPM Event and Semantic data model data for this process.

Often needed during testing and after initial deploys. Script needs to be run whenever the staging table is truncated to avoid duplicate conflicts in the data model tables.

Example: BPM/NYEC/ProcessStateReview/createdb/truncate\_data\_model\_NYEC\_PROCESS\_STATE\_REVIEW.sql

Implementation Steps:

Copy example and use as a template to create file:

BPM*/{project}* */{process}* /createdb/truncate\_data\_model\_*{project}*\_*{process}*.sql

TBA

# Test

TBA

# Install

Install Oracle SW, setup a database and populate it with Maxdat data objects.

PC Prerequisites:

* at least 4 GB of RAM (8 GB recommended)

## Oracle Software

Create a free user account on Oracle if you don’t already have on. Need to download.

<https://login.oracle.com/mysso/signon.jsp>

[Sign Up](https://login.oracle.com/mysso/%20https:/myprofile.oracle.com/EndUser/faces/profile/createUser.jspx?nextURL=https%3A%2F%2Flogin.oracle.com%2Fpls%2Forasso%2Forasso.wwsso_app_admin.ls_login%3FSite2pstoreToken%3Dvfor a free Oracle Web account

Go to this URL in a browser on you PC to download software:

<http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html>

Download Oracle software

* Accept License Agreement
* Oracle Database 11g Release 2 Standard Edition, Standard Edition One, and Enterprise Edition
* 11.2.0.1.0 or latest Oracle version with Microsoft Windows (x64)
* Microsoft Windows (x64)
  + Download files and save somewhere:

(each file is about 1GB and takes about 15 minutes each to download from home)

* + - select File 1
    - select File 2

Uncompress zip files saved above:

win64\_11gR2\_database\_1of2.zip -> win64\_11gR2\_database\_1of2

win64\_11gR2\_database\_2of2.zip -> win64\_11gR2\_database\_2of2

Copy files into single directory:

win64\_11gR2\_database\_1of2 -> win64\_11gR2\_database

win64\_11gR2\_database\_2of2 -> win64\_11gR2\_database

Run Oracle setup: (installs software)

win64\_11gR2\_database/setup.exe

* Configure Security Updates:
  + Email: (leave blank)
  + uncheck “I wish to receive security updates via My Oracle Support”
  + My Oracle Support Password: (leave blank)
  + -> Next
  + “Do you wish to remain uniformed of critical security issues in your configuration?”
    - Yes
* “Select and of the following install options.”
  + Create and configure a database.
  + -> Next
  + -> Finish
* Desktop Class
  + -> Next
* Perform full Database installation with basic configuration
  + (use defaults)
  + enter Administrative password (your choice of DBA password for your new database)
  + enter same to Confirm Password
  + -> Next
  + -> Finish

Oracle Software install occurs…..

* Windows Security Alert
  + Allow Access(Private Networks)
  + Allow Access

## Oracle Database

Repeat for each database that you are setting up. There should be one database per project (examples: ILEB, NYEC, etc.) that you are working on.

Start -> All Programs -> Oracle – OraDb11g – home 1

-> Configuration and Migration Tools -> Database Configuration Assistant

Select the operation that you wish to perform:

+ Create a Database

-> Next

Database Templates:

+ General Purpose of Transaction processing

-> Next

Database Identification:

Use this naming scheme to support multiple databases on a PC so that the database names are unique across the whole team.

Global Database Name: {project}{md}{developer\_initials}.maximus.com

SID: : {project}{md}{developer\_initials}

(‘md’ = Maxdat Development db)

Example: project = NYEC, developer initials = rk,

Global Database Name: nyecmdrk.maximus.com

SID: nyecmdrk

-> Next

Management Options:

+ Configure Enterprise Manager

+ Configure Database Control for local management

-> Next

Database Credentials:

+ Use the Same Administrative password for All Accounts

-> Next

Database File Locations:

Storage Locations:

+ Use Database File Locations from Template

-> Next

Recovery Configuration:

+ Specify Flash Recovery Area

(use defaults)

-> Next

Database Content

-> Next

Initialization Parameters:

+ Typical

(use defaults)

-> Next

-> Finish

(creates database, this may take an hour)

## Create Connections to System and Developer Users.

Set TNS\_ADMIN Windows system environment variable to point to directory containing the tnsnames.ora file:

Start -> Control Panel -> System -> Advanced system settings -> Environment Variables –> System variables

-> New

Variable name: TNS\_ADMIN

Variable value: {directory}

Example: C:\Development\Oracle

OK

Make sure that your tnsnames.ora file has a definition for your new database:

Example:

*NYECMDRK =*

*(DESCRIPTION =*

*(ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))*

*(CONNECT\_DATA =*

*(SERVER = DEDICATED)*

*(SERVICE\_NAME = nyecmdrk.maximus.com)*

*)*

*)*

**You must restart Oracle SQL Developer so that the tnsnames.ora changes are usable.**

In Oracle SQL Developer or other tool create connections to your new database for this user:

* sys - set Role: SYSDBA

## Maxdat Tablespaces and Users

Run these below as Oracle DBA user **sys** with role **sysdba**.

Create Maxdat tablespaces and users.

* Open file **svn://rcmxapp1d.maximus.com/maxdat/BPM/Core/createdb/create\_maxdat\_users.sql**
* Edit these lines in the file and customize for your user and project:

*Change the values for these 5 variables for each local database deploy.*

*v\_project\_initials varchar2(4) := 'nyec';*

*v\_developer\_initials varchar2(2) := 'rk';*

*v\_employee\_id varchar2(6) := '50472';*

*v\_maxdat\_user\_password varchar2(30) := 'maxdat\_changeme123';*

*v\_developer\_user\_password varchar2(30) := 'developer\_changeme123';*

* Run edited **create\_maxdat\_users.sql**

Install USER\_LOCK package:

* Download and run this script:

svn://rcmxapp1d.maximus.com/maxdat/BPM/Core/createdb/userlock.sql

In Oracle SQL Developer or other tool create connections to your new databases for this user:

Example:

* + {developer user} - example: rk50472

## Core ETL and BPM Data Objects

See svn://rcmxapp1d.maximus.com/maxdat/BPM/Core/doc /install\_core\_README.txt

for install instructions for Core ETL and BPM data objects.

## Per Project Deployed

Run these files below in your new database as the developer with proxy login to maxdat:

*Example: {project) – examples: Corp, NYEC, ILEB. TXEB*

BPM/{*project}*/createdb/populate\_BPM\_PROJECT\_LKUP.sql

BPM/*{project}*/createdb/populate\_BPM\_D\_OPS\_GROUP\_TASK.sql

BPM/{*project}*/createdb/MASH\_views.sql

## Per Process Deployed

Follow deploy instructions in each process’ **BPM/{***project}***/***{process}***/doc/{***project}***\_***{process}***\_README.txt** file.

Example:

BPM/Corp/ManageWork/doc/corp\_ManageWork\_README.txt

# Administration

## MAXDAT\_ADMIN Package

MAXDAT\_ADMIN is a BPM package installed on all MaxDat databases. It used primarily for lead users to configure the BPM and ETL processing on UAT and Production systems. It can also be used on development and local PC systems by all MAXDATdevelopers as a shortcut. All usage of this package is logged by username on the MAXDAT\_ADMIN\_AUDIT\_LOGGING table and any runtime errors are stored in the BPM\_LOGGING table.

To run this package on UAT or Production systems you must be authorized by your Manager and configured by the DBA. The DBA does this:

*“Grant execute on the MAXDAT\_ADMIN package directly to the requested users and don't use the PFE role we have since that would grant execute on all packages instead.”*

No need for extra configuration on development machines.

Available procedures:

COMPILE\_PROGRAM\_UNIT

CONFIG\_CALC\_JOB

CONFIG\_ETL

CONFIG\_JOB\_CONTROL

CONFIG\_QUEUE\_JOB

DISPLAY\_DBMS\_SCHEDULER\_JOBS

FIX\_JOBS

GATHER\_TABLE\_STATS

RESET\_BPM\_QUEUE\_ROWS

RESET\_BPM\_QUEUE\_ROWS\_BY\_BSL\_ID

RESET\_BPM\_QUEUE\_ROWS\_BY\_PBUEQ

SHUTDOWN\_JOBS

STARTUP\_JOBS

STOP\_DBMS\_SCHEDULER\_JOB

### Recompile a program unit natively.

**COMPILE\_PROGRAM\_UNIT**

(p\_program\_unit\_type in varchar2,

p\_program\_unit\_name in varchar2)

Program Unit must be one of these types:

‘FUNCTION’

‘PACKAGE’

‘PROCEDURE”

Example:

execute MAXDAT\_ADMIN.COMPILE\_PROGRAM\_UNIT(‘PACKAGE’,’NYEC\_PROCESS\_APP’);

### Configure a BPM calculation job. Enable or disable a job.

**CONFIG\_CALC\_JOB**

(p\_package\_name in varchar2,

p\_procedure\_name in varchar2,

p\_process\_enabled in varchar2)

Updates the PROCESS\_BPM\_CALC\_JOB\_CONFIG table used by the BPM Process Control Job to determine which BPM calculation jobs to start or stop.

Process Enabled must be either: ‘Y’ (enable job) or ‘N’ (disable job)

Example:

execute MAXDAT\_ADMIN.**CONFIG\_CALC\_JOB**('NYEC\_PROCESS\_APP','CALC\_DNPACUR','Y');

### Configure ETL processing.

**CONFIG\_ETL**

(p\_name in varchar2,

p\_value in varchar2)

Updates the CORP\_ETL\_CONTROL table used to configure ETL processing.

Name must be an existing CORP\_ETL\_CONTROL.NAME value.

Example:

execute MAXDAT\_ADMIN.**CONFIG\_ETL**(‘STATE\_REVIEW\_LOOK\_BACK\_DAYS','5');

### Configure the BPM job controller.

**CONFIG\_JOB\_CONTROL**

(p\_parameter\_name in varchar2,

p\_parameter\_value in varchar2)

Updates the PROCESS\_BPM\_QUEUE\_JOB\_CTRL\_CFG table used to configure the BPM job controller package PROCESS\_BPM\_QUEUE\_JOB\_CONTROL.

Parameter Name must be one of these:

'MAX\_TOTAL\_NUM\_JOBS'

'NUM\_JOBS\_TO\_DEL\_DURING\_ADJUST'

'NUM\_JOBS\_TO\_ADD\_DURING\_ADJUST'

'NUM\_GROUP\_CYCLES\_BEFORE\_ADD'

'CONTROL\_JOB\_SLEEP\_SECONDS'

‘START\_DELAY\_SECONDS'

'STOP\_DELAY\_SECONDS'

'PROCESSING\_ENABLED' (Parameter Value must be either ‘Y’ or ‘N’)

Example:

execute MAXDAT\_ADMIN.**CONFIG\_JOB\_CONTROL**('CONTROL\_JOB\_SLEEP\_SECONDS','30');

### Configure BPM queue processing.

**CONFIG\_QUEUE\_JOB**

(p\_bsl\_id in number,

p\_bdm\_id in number,

p\_parameter\_name in varchar2,

p\_parameter\_value in varchar2);

Updates the PROCESS\_BPM\_QUEUE\_JOB\_CONFIG table used to configure the BPM queue job package PROCESS\_BPM\_QUEUE.

Parameter Name must be one of these:

'MIN\_NUM\_JOBS'

'INIT\_NUM\_JOBS'

'MAX\_NUM\_JOBS'

'BATCH\_SIZE'

'ENABLED' (Parameter Value must be either ‘Y’ or ‘N’)

Example:

execute MAXDAT\_ADMIN.**CONFIG\_QUEUE\_JOB**(4,1,'MAX\_NUM\_JOBS','6');

### Display DBMS Scheduler job names.

**DISPLAY\_DBMS\_SCHEDULER\_JOBS**

Displays a list of job DBMS Scheduler job names from ALL\_SCHEDULER\_JOBS.JOB\_NAME where OWNER is ‘MAXDAT’. View dbms\_output to see the list of job names.  
SQL Developer -> View Dbms Output -> + (on Dbms Output window) -> select database -> OK

Example:

execute MAXDAT\_ADMIN.**DISPLAY\_DBMS\_SCHEDULER\_JOBS**;

### Fix metadata of incorrect BPM queue jobs.

**FIX\_JOBS**

Fix metadata of defective jobs with null parameters and stop them.

Fix metadata of jobs that are not running.

Example:

execute MAXDAT\_ADMIN.**FIX \_JOBS**;

### Gather table statistics.

**GATHER\_TABLE\_STATS**

(p\_owner in varchar2,

p\_table\_name in varchar2,

p\_degree in number);

Gather table statistics. Helps improve performance.

Updated statistics will not be used by currently running programs.

Stop and restart program to be able to use the updated statistics.

Degree is the number of parallel processes to run when gathering statistics.

Example:

execute MAXDAT\_ADMIN.**GATHER\_TABLE\_STATS**('MAXDAT','BPM\_UPDATE\_EVENT\_QUEUE',8);

### Reset all BPM processing queue rows that were reserved but failed to process.

**RESET\_BPM\_QUEUE\_ROWS**

Reset queue rows that are blocked, mostly likely due to previous processing error, so that they can be processed.

Sets BPM\_UPDATE\_EVENT\_QUEUE.PROCESS\_BUEQ\_ID to null where PROCESS\_BUEQ\_ID is not null.

Example:

execute MAXDAT\_ADMIN.**RESET\_BPM\_QUEUE\_ROWS**;

### Reset BPM processing queue rows that were reserved but failed to process by BPM Source Lookup ID.

**RESET\_BPM\_QUEUE\_ROWS\_BY\_BSL\_ID**

(p\_bsl\_id in number)

Reset queue rows that are blocked, mostly likely due to previous processing error, so that they can be processed.

Sets BPM\_UPDATE\_EVENT\_QUEUE.PROCESS\_BUEQ\_ID to null for a process BSL\_ID where PROCESS\_BUEQ\_ID is not null.

Example:

execute MAXDAT\_ADMIN.**RESET\_BPM\_QUEUE\_ROWS\_BY\_BSL\_ID**(2);

### Reset BPM processing queue rows that were reserved but failed to process by BPM Queue PROCESS\_BUEQ ID.

**RESET\_BPM\_QUEUE\_ROWS\_BY\_PBUEQ**

(p\_process\_bueq\_id in number)

Reset queue rows that are blocked, mostly likely due to previous processing error, so that they can be processed.

Sets BPM\_UPDATE\_EVENT\_QUEUE.PROCESS\_BUEQ\_ID to null for a PROCESS\_BUEQ\_ID (a job run batch reservation ID) where PROCESS\_BUEQ\_ID is not null.

Example:

execute MAXDAT\_ADMIN.**RESET\_BPM\_QUEUE\_ROWS\_BY\_PBUEQ**(734);

### Shutdown all BPM control, queue and calculation jobs.

**SHUTDOWN\_JOBS**

Stop the BPM job controller, stop all queue jobs, repair any invalid queue jobs (FIX\_JOBS), stop all calculation jobs.

Example:

execute MAXDAT\_ADMIN.**SHUTDOWN\_JOBS**;

### Startup all BPM control, queue and calculation jobs.

**STARTUP\_JOBS**

Start the BPM job controller (which starts queue jobs) and start calculation jobs.

Example:

execute MAXDAT\_ADMIN.**STARTUP\_JOBS**;

### Stop a DBMS Scheduler job.

**STOP\_DBMS\_SCHEDULER\_JOB**

(p\_job\_name in varchar2)

Job Name is the same ALL\_SCHEDULER\_JOBS.JOB\_NAME listed by DISPLAY\_DBMS\_SCHEDULER\_JOBS.

Example:

execute MAXDAT\_ADMIN.STOP\_DBMS\_SCHEDULER\_JOB('PROCESS\_Q\_BY\_BSL\_352');

## Local Personal Oracle Database on a Windows PC

### Oracle Listener

The Oracle listener handles incoming client requests and manages the traffic between it and the database.

Run the Oracle listener command line interface:

(Windows) Start -> All Programs -> Accessories > Command Prompt ->

Type:

lsnrctl

(Type “exit” to exit and close the command prompt when finished.)

#### Status

LSNRCTL> status

If your database is already started you should see it listed in the **Services Summary**.

#### Start

LSNRCTL> start

#### Stop

LSNRCTL> stop

#### Exit

LSNRCTL> exit

### Oracle Database

(Windows) Start -> All Programs -> Oracle OraD\_11g\_home1 -> Configuration and Migration Tools

-> Administration Assistant for Windows

-> Oracle Managed Objects -> Computers -> *{computer\_name}* -> Databases

-> (right-click) on database

#### Start

-> Start Service

#### Stop

> Stop Service

## Manually Stop Jobs

Normally you stop all BPM control, calculation and queue jobs via this command.

execute PROCESS\_BPM\_QUEUE\_JOB\_CONTROL.SHUTDOWN\_JOBS;

If that fails you can manually stop jobs by running these commands:

### List running DBMS Scheduler jobs.

select JOB\_NAME from ALL\_SCHEDULER\_JOBS where OWNER = 'MAXDAT' order by JOB\_NAME asc;

### Stop a running DBMS Scheduler job.

(repeat for each running job)

execute dbms\_scheduler.stop\_job( *{job\_name}* );

Example:

execute dbms\_scheduler.stop\_job('PROCESS\_Q\_BY\_BSL\_5112');

Stop the PROCESS\_BPM\_QUEUE\_CONTROL job first as it can cause new jobs to be created.

The PROCESS\_Q\_BY\_BSL\_\* jobs can be stopped next.

The CALC\_D\*CUR jobs can be stopped last as they usually only run for a few minutes immediately after the database’s midnight.

Any ‘PROCESS\_Q\_BY\_BSL\_\*’ jobs stopped in this way will leave the queue job metadata tables

in an incorrect state. Restarting all the jobs

execute PROCESS\_BPM\_QUEUE\_JOB\_CONTROL.STARTUP\_JOBS;

will cause any incorrect metadata to be automatically fixed.

If it will be some time before the jobs are restarted you can manually fix the job metadata by running this command:

execute PROCESS\_BPM\_QUEUE\_JOB\_CONTROL.FIX\_JOBS;

# Glossary

**BPM** – Business Process Management

**Canonical Database** – The central repository that holds lookup definitions for all MaxDat projects and processes. Individual project databases will only hold shared or project-specific definitions. Currently this is the MAXDTDEV db.

**Instance** – In MaxDat an instance is a path through a business process flow chart, made by a particular client, such as one client applying for benefits. This translates into one row in the final staging table in the database that describes the detailed state of that process for the client. This is referred to in the BPM Event data model in the BPM\_INSTANCE table.

MicroStrategy – Business intelligence and reporting software.

**Process** – A business process described by a requirement spreadsheet. Example: Process Letters

**Project** – A set of business processes used to fulfill contractual obligations for BPM for an entity such as a state benefit program. Example: Illinois Enrollment (ILEB)

**Queue Archive Table** – BPM\_UPDATE\_EVENT\_QUEUE\_ARCHIVE – Stores Queue Table rows that have been processed.

**Queue Table** – BPM\_UPDATE\_EVENT\_QUEUE – Stores XML representation of selected staging table columns with before and after insert/update states. Processed later to populate the BPM Event and Semantic data models.

**SVN** – Subversion – Software revision control system.

**TBA** –To Be Added – Information to be added later.