Code guidelines

We begin by adopting applicable portions of the PSI document *Naming Conventions and Coding Guidelines.*

Naming conventions

PL/SQL Packages

When new functionality is implemented, new packages shall be used. The name consists of three parts, a prefix, the main part and the suffix. Overall, the package name shall not be longer than 30 characters. Package names are written in camel case and underscores are not allowed to be used.

The prefix defines the scope of the package:

- **tab**: automatically generated packages for each table providing functions for table operations (i.e. select, update or exists).
- **ent**: packages intended to only work on a single entity, usually tables, but could be other objects. Should not call packages having a bigger scope (i.e. kmp). Could contain functions to classify entries in the related table, such like "isTransitionSlab".
- **kmp**: most functionality for business use cases should be written in these packages.
- **dlg**: functions to be called by the GUI – the GUI should never call any other packages directly.
- **tlg**: generated for all interfaces to define message structures in PL/SQL.
- **glo**: used for globally used functionality, that extend over multiple entities or business use cases. Often used for i.e. master data definitions. Should not call kmp packages.

The main part of the name should be descriptive enough to be understandable on a glance. Common abbreviations can be used, but we should not invent new abbreviations just to keep the name short. I.e. "QMSamplePlanning" is a descriptive name, but "QMSP" would probably not be understood.

Tables

All table definitions must be done globally; they are always valid for all divisions. When a new table is needed, it should get the prefix "**NUCOR_**".

Again, the table name should be descriptive to understand what the table contains. In addition, the table name shall not be longer than 30 characters. Table names are always written in upper case. The usage of underscores should be prevented.

When tables are related (i.e. for detail tables), the relation should be clearly visible in the naming of the tables.

Domain / Value List defining tables should start with the prefix "**NUCOR_VL**".

Each project specific table should have a table comment starting with "**PRJ:**".

An example for a global project specific table could be "NUCOR_QMSAMPHEATTREATCYCLE".

Table Columns

When columns are added to standard tables, the new column shall have the prefix X. This will always be done globally; the new columns are always valid for all divisions. They should be written in upper case and not longer than 30 characters. The use of underscores should be prevented. Each project specific column should have a column comment starting with "PRJ:".

When a project specific column holds only a yes/no flag, the prefix shall be **XFLAG**.

For columns of project specific tables, no prefix is needed.

An example for a project specific column would be "XFLAGTRANSITION-SLAB".

Views

Project specific views should get a prefix depending on if it is a division specific or a global view. Each view has a suffix:

- **_V**: used for internal views, not shown in the GUI. These views should only contain the necessary information and must be performance optimized, when used in PL/SQL code.
- **_VD**: used for views shown in the GUI. These views should not be used in PL/SQL code. If it is necessary to use it in the GUI and in PL/SQL code, a _V view needs to be created and the _VD view should extend it by the additional GUI fields.

As before, the view name should be descriptive to understand what the view shows. In addition, the view name shall not be longer than 30 characters. View names are always written in upper case. The usage of underscores should be prevented (except for the prefix and suffix).

An example for a global project specific view is "NGLO_QMSAMPHEATTREATCYCLE_VD". This one would be used in the Office GUI or PDA.

Others

This chapter describes naming conventions of other PL/SQL objects. When not stated otherwise, everything should be written in camel case and not longer than 30 characters.

Functions / Procedures should start with a lower letter and describing the action the function / procedure is doing. Local functions and procedures (which are not defined in the package specification) shall add a prefix "L_" and should be either in the beginning of the package when they are used multiple times or directly before the procedure calling it, when it is only used once.

Parameters of functions and procedures shall have a prefix before the descriptive name as follows:

- **pi_** for input parameters ****
- **po_** for output parameters
- **pio_** for parameters shared with the calling function / procedure (used as input and output).

Variables must have a prefix "v" before the descriptive name. The only exception to that is for counting variables in loops, which shall be named "i" (or "j", "k", etc. for nested loops).

Lists / Variable Tables should have the prefix "vt".

Constants shall be written in upper case without prefix.

Indexes must have the division prefix, followed by the table name and then the column name (or a good name for grouping, if it is more than one), having the suffix IDX. The different parts should be separated by an underscore i.e. NGLO_MAT_SPERRGRUND_IDX.

Sequences shall be defined globally whenever possible and must have the division prefix. This is followed by the sequence name and should end with the suffix SQ. The different parts should be separated by an underscore i.e. NGLO_TICKETNO_SQ.

Constraints shall be defined globally whenever possible and must have the division prefix. This is followed by FK and finished with the table relation for foreign key constraints or followed by the table name and the column name ending with the suffix CHK for check constraints. The different parts should be separated by an underscore i.e. NGLO_FK_APPOINT2PG for a foreign key constraint and NGLO_MAT_XFLAGHOLD_CHK.

Generic Actions shall be named with a division prefix and then a description what this action is doing. The use of more than this one underscore to separate the prefix shall be prevented. An example would be NGLO_CREATELOADS.

Style conventions

Handling of Prj Packages

Prj packages shall be changed as less as possible. If code needs to be added, it should only call a project specific package and all new code should be there.

All necessary changes in there should be marked as follows:

- Code Addition: Add "«<PROJECT SPECIFIC BEGIN»>" and "«<PROJECT SPECIFIC END»> before and after the code addition.
- Code Deletion: Do not delete the code but comment the whole block and a comment "«<PROJECT SPECIFIC NOT NEEDED»>" before it.
- Code Change: When single lines are changed (i.e. to assign different values), a comment "«<PROJECT SPECIFIC CHANGED»> at the end of the line.

This will help a lot in later release upgrades.

PL/SQL Guidelines

- All used identifiers should be clearly understandable and the intent of a code line should be clear without adding comments. When this is not possible and a code line is not immediately understandable, a precise comment shall be added.
- All functions / procedures should have the standard comment block above it, describing in one to two sentences, what the function does. If the function is more complicated, a longer text can be used and the function parameters should be explained.
- An indent of two spaces shall be used in each IF, LOOP and EXCEPTION block.
- Oracle reserved words should be written in upper case (i.e. IF, SELECT, BEGIN, etc.).
- For selection, updating, deleting or checking existence of a row, use the related tab package whenever possible.
- A code line should generally not be longer than 80-100 characters. Of course, this is a soft rule, but it makes code much more readable if this rule is followed.
- Functions / Procedures shall not be longer than about one screen height (~100 lines). Again, this is a soft rule, but when procedure are too long it will be hard to understand and debug them. Consider the use of local functions to split up the function into several sub-parts.
- Lists must start by 1 and shall not have any gaps. However, be careful when going through a loop, since the standard does not always starts by 1.
- Each function should write logs as needed. Set Log Levels carefully, information for debugging should be level 7-9. All loggings should be clearly understandable, so do not just log a single variable.

- Consider to move SELECT statements into separate functions, maybe even to a separate ent package. Thus you prevent writing the same statement multiple times, if it is needed somewhere else, too.
- Make sure all SQL queries are performant and use BULK collect whenever possible to avoid context switching. Consider the use of Indexes.

Table Definitions

Each new table definition should be bundled in one SQL file, including DDL comments and constraints. It should allow multiple executions of the script without losing data. This means, instead of a DROP TABLE in the beginning of the script, it should check for the existence of the table. When new columns are needed to be added, consider using Add Column statements.

Each table should also contain the audit columns DTCREATED, USERIDCREATED, DTUPDATED, USERIDUPDATED, UPDATECOUNTER and DTARCHIVED including a script to generate the triggers for these columns.

Additionally, all tables containing master data shall have a SYSTEM column.

The script should also include the standard DD generation.

For each table, only one extension script shall be used and extended. The script should be written in a way, which allows multiple executions of it.

View Definitions

All views should always contain a script for the data dictionary generation. Make sure, that all used tables are mentioned in that script as well as the DDSYSTEM. In case for Nucor specific views the DDSYSTEM can be: - **NGLO** For NGLO views - **PRJ** For division specific views

Make sure, the view is performant, i.e. by avoiding full table scans and adding indexes to a table, where needed. A good way to check this is by using `Explain Plan`.

Important: For flexibility and effectiveness views need to follow these additional guidelines: - No view shall be made global (NGLO_XXX_VD) unless it belongs to the global application functionality (eg: Gatehouse views, OP PDA views, etc). - This means that for the global views, any change will be propagated to all divisions equally. - Note that all views belonging to the global scope should not contain standard views. - Modified standard views must be kept division specific - Reasons for standard views to not be kept globally: - It could get messy if a modification becomes rather division specific than globally. - Gives us the flexibility to adapt changes division specific - Does not add more complexity to all divisions if one decides to add a complex join - If a division needs to copy a view (either modified standard or project specific) from another division, that view must be brought over to the respective project ADO GIT repository and renamed appropriately using our view naming convention guidelines.

Master Data & DD Configuration

Master data & DD Configuration should be flagged as project specific. For master data, the SYSTEM should be NGLO for global and PRJMD for Division specific master data (Eg: Prod lines, VL tables, GloHandlerList records, etc). For DD Configuration, the DDSYSTEM columns should NGLO if it is global configurations or PRJ if it is a Division specific configuration (i.e. DD Objects, DD Domains, Mapping Rules, DD_SAP). Additionally, we have introduced special DDSYSTEMs for special cases: - NGLO-OD – For any DD metadata related to the PSImetals OD GUI - NGLO-PSIWEB – For any DD metadata related to the Global PSImetals Web app - PRJ-PSIWEB – For any DD metadata related to division specific PSImetals Web app