

**SAP ABAP Programming Standard**

Version – 1.30

|  |  |
| --- | --- |
| **Document Information** | |
| Title: | *Goodyear ABAP Development Standards* |
| Key words: | *ABAP/4 Program* |
| Author: |  |
| File Name: | *Goodyear ABAP Development Standards* |
| Path: |  |
| Created: |  |
| Last Edited: |  |
| Number of pages: |  |

.

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Vision Date** | **Revision Description** | **Author** |
| *1.00* | *01-Aug-2020* | *Initial Release of Global Dev Standards* | *Raphael Donor* |
| *1.1* | *16-Nov-2020* | *Modification with Alignment to NA* | *Mon Magallanes* |
| *1.2* | *11-Dec-2020* | *Modification with Alignment to NA* | *Mon Magallanes* |
| *1.3* | *18-Jan-2021* | *Modification with Alignment to NA* | *Mon Magallanes* |

**TABLE OF CONTENT**

1. INTRODUCTION 6

2. GLOBAL DEFINITION 7

2.1 SAP OBJECT TYPES 7

2.2 APPLICATION AREAS / MODULES 7

2.3 PROGRAM TYPES 8

3 NAMING CONVENTIONS 9

3.1 ABAP/4 PROGRAM 9

3.1.1 Executable Program 9

3.1.2 SAPscript / Style 9

3.1.3 SMARTFORMS / SMARTSTYLES / Text Modules 9

3.1.4 Adobe Forms / Interface 10

3.1.5 Program Includes 10

3.2 MODULE POOL PROGRAMS 11

3.2.1 Module Pool Programs 11

3.2.2 Include Programs 11

3.2.3 Info Types Module Programs 12

3.2.4 Info Types Include Programs 12

3.2.5 Screen Painter PF Key Definitions 12

3.3 PROGRAM OBJECTS 13

3.3.1 Event 13

3.3.2 Subroutines 13

3.3.3 Process Before Output (PBO) Modules 13

3.3.4 Process After Input (PAI) Module 14

3.3.6 Screen Number 14

3.3.7 GUI Status 14

3.3.8 GUI Title 14

3.3.9 Transaction code 15

3.4 DICTIONARY OBJECTS 16

3.4.1 Tables 16

3.4.2 Structures 16

3.4.3 Table Types 17

3.4.4 Views 17

3.4.5 Table Indices 17

3.4.6 Table Fields 17

3.4.7 Data Elements 18

3.4.8 Domains 18

3.4.9 Lock Objects 19

3.4.10 Search Help 19

3.4.11 Type Groups 19

3.4.12 ABAP on HANA objects 19

3.4.12.1 CDS View 19

3.4.12.2 CDS Table Function 20

3.4.12.3 ABAP Managed Database Procedure (AMDP) 20

3.4.13 Restrictions for ABAP/4 Dictionary objects 20

3.5 FUNCTION GROUP OBJECTS 21

3.5.1 Function Group 21

3.5.2 Function Modules 21

3.6 ABAP Objects 22

3.6.1 Class 22

3.6.2 Interface 22

3.6.3 Exception Class 22

3.7 OTHER OBJECTS 23

3.7.1 Messages in SAP Programs 23

3.7.2 Development Classes / Package 23

3.7.3 SET/GET Parameters 23

3.7.4 Business Addins 24

3.7.5 Enhancements 24

3.8 Naming Convention Summary 25

4. CODING STANDARD CONVENTION 26

4.1 FORMATTING STANDARDS 26

4.1.1 Program Structures 26

4.1.2 Placement of Process Blocks 26

4.1.4 Program Documentation 27

4.1.5 Change Documentation 28

4.2 PRETTY PRINTER 30

4.2.1 Pretty Printer Configuration 30

4.2.2 Pretty Printer Format Execution 30

4.3 CHANGE OF STANDARD CODE CLONES 30

4.3.1 Naming Convention 30

4.3.2 Identifying changes 31

4.3.3 Change Tag when Deleting Codes 31

4.3.4 Change Tag when Inserting Code 32

4.4 CODING STANDARDS 32

**4.4.1** **Program Heading** 32

**4.4.2** **Field and Data Manipulations** 32

4.3.2.1 Selection-options/Parameters 32

4.3.2.2 Data 32

4.3.2.3 Variable Typing 33

4.3.2.4 TYPE 33

4.3.2.5 CASE 33

4.3.2.6 Internal Fields for Constant 33

**4.4.3** **Authorization Checking** 33

**4.4.4** **Multi-level IF Statements** 33

**4.4.5** **Standard Functions** 33

**4.4.6** **Return Codes** 33

**4.4.7** **Internal Tables** 33

**4.4.8** **Field Symbols** 33

**4.4.9** **Text Symbols** 34

**4.4.10** **Subroutines** 34

**4.4.11** **CHECK, EXIT, RETURN** 34

**4.4.12** **AT PFnn** 34

**4.4.13** **EXPORT TO MEMORY** 34

**4.4.14** **Attributes** 34

**4.4.15** **User Interface (GUI)** 34

**4.4.16** **Native SQL** 35

**4.4.17** **Data Update** 35

**4.4.18** **Online Help** 35

**4.4.19** **Legacy System Programs** 35

**4.4.20** **Object Oriented Programming** 35

4.5 GENERAL STANDARDS 36

**4.5.1** **Correction and Transport System** 36

**4.5.2** **Batch Input Program Designs** 36

**4.5.4** **Module Pools** 36

**4.5.5** **Altering an Active Objects** 37

**4.5.6** **Function Groups** 37

**4.5.7** **Maintenance/Correction of Production Codes** 37

**4.5.8** **Use Include Modules** 37

**4.5.9** **Country Specific Codes** 37

5. MODIFYING SAP PROGRAMS 41

5.1 NAMING CONVENTIONS 41

6. PERFORMANCE GUIDELINES 41

6.1 METHODS OF IMPROVING GENERAL PERFORMANCE 41

7. SAP QUERY 42

7.1 QUERY AREA 42

7.2 USER GROUPS 43

7.3 INFOSETS 43

7.4 QUERY 43

7.5 AUTHORIZATIONS IN QUERY 43

8. MISCELLANEOUS 44

8.1 ABAP Program Data – General 44

8.2 Web Dynpro Naming Convention 45

# 1. INTRODUCTION

This document contains the standards to be followed by Goodyear ABAP Development Team to SAP R/3 implementation and other Goodyear SAP R/3 rollout projects for developing and maintaining SAP applications. The objectives of the standards are to:

1. Provide consistency to facilitate maintenance.

2. Track changes to support upgrades and releases.

3. Develop software that performs efficiently.

The document is divided into several sections. With the exception of the section ‘Guidelines for Performance’, all standards are requirements and must be adhered to. The ‘Guidelines for Performance’ section has been developed to assist you in making decisions on how to write efficient code and the ideas presented are only guidelines.

**It is the responsibility of all developers to follow these standards.**

This is a living document.

Some of these standards documented here will change and new ones will be added.

New standards and changes to standards should be communicated to the owner of this document.

# 2. GLOBAL DEFINITION

## SAP OBJECT TYPES

**A** Authorization Object

**B** Authorization Profile

**BD** BADI

**C** CDS View

**CL** Class

**CX** Exception Class

**D** Domain

**E** CDS Table Function

**EI** Enhancement Implementation

**F** Function Module

**G** Function Group

**GM** Function Group for Table Maintenance

**I** Table Index

**IF** Class Interface

**IM** BADI Implementation Class

**J** Background Job Names

**K** SET/GET Parameters

**N** Type Group

**P** Programs for Production

**Q** Report Variants

**S** Structure

**SH** Search Help

**T** Tables

**TT** Table Types

**V** View

**X** Data Element

## APPLICATION AREAS / MODULES

Application Areas

**A** Fixed Asset Management

**C** PPC

**D** DASS (Control Station)

**E** RIVA

**F** Financial Accounting

**G** General Ledger

**H** Human Resource Planning

**I** Plant Maintenance

**J** Publishing

**K** Cost Accounting

**L** Warehouse Management

**M** Materials Management

**N** Hospital

**P** Human Resource

**Q** QSS (Quality Assurance)

**R** CRM system

**S** Basis

**U** EDM (Enterprise Data Model)

**V** Sales and Distribution

**W** MSS (Material Merchandising System)

**Z** Cross Application

Modules

AC Accounting General

BC Basis

BW Business Warehouse

CFN Central Finance

CA Cross-Application Components

CO Controlling

CRM Customer Relationship Management

CS Customer Service (formerly: PS-SM)

EC Enterprise Controlling

EP Enterprise Portal

FI Financial Accounting

FOS Fleet Online Solution (Goodyear)

GRC Governance, Risk and Compliance

LE Logistics Execution

LO Logistics General

MM Material Management

MDG Master Data Governance

PM Plant Maintenance

PP Production Planning

QM Quality Management

SD Sales and Distribution

SRM Supplier Relationship Management

TR Treasury

XX Miscellaneous

OO Cross/Common Objects

## PROGRAM TYPES

**C**  Conversion

**N**  Interface

**J**  Job

**I**  Include

**R**  Report

**U** Update

**X** Extract

# 3 NAMING CONVENTIONS

## *3.1 ABAP/4 PROGRAM*

### 3.1.1 Executable Program

Executable program is a program which can be started without a transaction code,  
either directly or in the background. Depending on the requirements the program may update database tables, create a batch-input session or produce a report. Report programs are generally independent programs, which are not assigned to a module pool.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | |
| **YY** – Module (see table 2.2) | | | |
| **\_** – Separator | | | |
| **<desc>** - Free text (description) | | | |
| **Ex. ZFI\_CREDIT\_REPORT** | | | |

### 3.1.2 SAPscript / Style

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Format** | **Z** | **SS** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | |
| **SS** – Program type – SAP Script (fixed) | | | |
| **<desc>** - Free text (description) | | | |
| **Ex. ZSS\_POFORM** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Format** | **Z** | **SS** | **<desc>** |
|  | **Z** – Namespace (fixed) | | |
| **SS** – Program type - Style (fixed) | | |
| **<desc>** - Free text (description) | | |
| **Ex. ZSSPOFRM** | | |

### 3.1.3 SMARTFORMS / SMARTSTYLES / Text Modules

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **SF** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **SF** – Program type – Smartforms (fixed) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZFISF\_PAYFORM** | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **SS** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **SS** – Program type – Style (fixed) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZFISS\_PAYFORM** | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **TM** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **TM** – Program type – Text module (fixed) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZFITM\_PAYFORM\_TXT1** | | | | |

### 3.1.4 Adobe Forms / Interface

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **AF** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **AF** – Program type – Adobe Forms / Interface (fixed) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZFIAF\_PAYFORM** | | | | |

### 3.1.5 Program Includes

Any ABAP/4 “Include” program will follow the standards for ABAP/4 program names and will be a program type “I” in the attribute field.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **\_** | **<desc>** | **\_** | **<type>** |
|  | **Z** – Namespace (fixed) | | | | | |
| **YY** – Module (see table 2.2) | | | | | |
| **\_** – Separator | | | | | |
| **<desc>** - Free text (description) | | | | | |
| **<type> -** Include type **TOP** – Data declaration **FXX** – Routine include (00 to 99) **CLS** – Class  **SCR** – Screen | | | | | |
| **Ex. ZFI\_CREDIT\_REPORT\_F01** | | | | | |



## *3.2 MODULE POOL PROGRAMS*

Module pool is [ABAP program](javascript:call_link('abenabap_program_glosry.htm')" \o "Glossary Entry) that normally contains [dynpros](javascript:call_link('abendynpro_glosry.htm')" \o "Glossary Entry) and [dialog modules](javascript:call_link('abendialog_module_glosry.htm')" \o "Glossary Entry), and is started using [transaction codes](javascript:call_link('abentransaction_code_glosry.htm')" \o "Glossary Entry).

An include program can contain several modules of the same type (such as only PBO modules or only PAI modules). The main program then consists of a sequence of INCLUDE statements that link the modules to the module pool.

Include programs improve the readability of programs and make maintenance easier. An include program is a separate program with 2 main functions:

1. It contains code that can be used by several programs (e.g. lengthy data declarations).
2. It helps to modularize programs, which consist of many different logically related parts. Each of these parts is stored as a separate include program

Dialogs/Screens, Subroutines, and Module pool have the same naming structure and are grouped and described below. Info Type module pool and include programs has a different naming structure, which is described separately.

### 3.2.1 Module Pool Programs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Format** | **SAPMZ** | **YY** | **\_** | **<desc>** |
|  | **SAPMZ** – Namespace (fixed) | | | |
| **YY** – Module (see table 2.2) | | | |
| **\_** – Separator | | | |
| **<desc>** - Free text (description) | | | |
| **Ex. SAPMZSD\_CREDIT\_CHK** | | | |

### Include Programs

Positions 1 – 6 of include program must be the same as positions 4 – 9 of the related module pool program.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Format** | **MZ** | **YY** | **\_** | **<desc>** |
|  | **MZ** – Namespace (fixed) | | | |
| **YY** – Module (see table 2.2) | | | |
| **\_** – Separator | | | |
| **<desc>** - Free text (description) | | | |
| **Ex. MZSD\_CREDIT\_CHK\_TOP** | | | |

While the descriptive text at the end of the Include program name is freely definable, there are cases where it must follow a specific naming convention. For PAI, PBO and Form modules, the 3 character descriptive text must commence with I, O or F respectively, followed by a 2 digit sequence number, commencing with 01 (e.g. I01). For the top level include program, the 3 character descriptive text must be TOP. As the system creates the include modules for these special cases the appropriate name is suggested and should not be altered.

### Info Types Module Programs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Format** | **MP** | **9** | **###** | **00** |
|  | **MP** – Namespace (fixed) | | | |
| **9** – Custom infotype indicator (fixed) | | | |
| **###** – Infor type number ( 000 – 999 ) | | | |
| **00** – Fixed | | | |
| **Ex. MP902000** | | | |

### Info Types Include Programs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Format** | **MP** | **9** | **###** | **YY** |
|  | **MP** – Namespace (fixed) | | | |
| **9** – Custom infotype indicator (fixed) | | | |
| **###** – Infor type number ( 000 – 999 ) | | | |
| **YY** – Sequence number increments ( 10 – 90 ) | | | |
| **Ex. MP902010** | | | |

### Screen Painter PF Key Definitions

PF Key assignments must follow the R/3 PF-key definition standards.

The assignments will vary depending on the status type (LIST, LIST IN DIALOG BOX, SCREEN, and DIALOG BOX) of the status that you create using Menu Painter (transaction SE41) but will follow these general guidelines:

PF 1 Help

PF 2 Choose

PF 3 Back

PF 4 Possible Entries

PF 10 Menu Bar

PF 11 Save

PF 12 Cancel

PF 13 Print

PF 14 Delete

PF 15 Exit

PF 20 Download

PF 21 First Page

PF 22 Previous Page

PF 23 Next Page

PF 24 Last Page

## *3.3 PROGRAM OBJECTS*

### Event

An event is an external or internal action that has taken place. At the time the event arises, it can trigger one or more functions, and can be the result of a function. An event causes the execution of a specific block of authorization checkic in a program. Events are associated with specific key words in the program; e.g., pressing a PF key (AT PF1), reading the database (GET), or indicating the end of a page (END-OF-PAGE).

|  |  |
| --- | --- |
| **Format** | **<desc>** |
|  | **Ex. BACKUP\_COMPLETE** |

### Subroutines

Subroutines are blocks of program code (modules) which can be called from ABAP/4 programs. Subroutines allow frequently used parts of a program or algorithms to be stored once.

There are two types of subroutines:

Internal subroutines: The source code of internal subroutines is in the same ABAP/4 program as the calling procedure (internal call).

External subroutines: The source code of external subroutines is in an ABAP/4 program other than the calling procedure (external call).

Internal subroutines are mainly used to modularize and structure individual programs. It is possible to create ABAP/4 programs, which contain only subroutines. These programs are not executable, but are used by other ABAP/4 programs as pools of external subroutines.

|  |  |
| --- | --- |
| **Format** | **<desc>** |
|  | **Ex. FIND\_ORDER** (Sales subroutine) |

### Process Before Output (PBO) Modules

The flow logic of a screen drives the processing sequence for that screen. The PBO (process before output) event is triggered just before any screen is displayed in a dialog program and is generally used to prepare the screen for output (e.g. initialize screen field data, suppressing screen fields, etc.).

The program code to be processed by each PBO event is entered in a special subroutine called the PBO module. The PBO event on each screen can trigger a separate PBO module. Alternatively, a single PBO module can be referenced by the PBO event on many screens. (Useful for performing a specific initialization on different screens).

|  |  |
| --- | --- |
| **Format** | **<desc>** |
|  | **Ex. DISPLAY\_MENU** (Show Screen Menu) |

### Process After Input (PAI) Module

The flow logic of a screen drives the processing sequence for that screen. The PAI (process after input) event is triggered when the user responds to a screen in a dialog program (usually involving data having been changed on the screen and a menu function, push button or function key having been selected). The PAI event is generally used to check the user input and to trigger appropriate dialog steps (e.g. the update task).

The program code to be processed by each PAI event is entered in a special subroutine called the PAI module. The PAI event on each screen can trigger a separate PAI module. Alternatively, a single PAI module can be referenced by the PAI event on many screens (useful for a specific validation to be done on different screens).

|  |  |
| --- | --- |
| **Format** | **<desc>** |
|  | **Ex. VALIDATE\_MATERIAL** (Materials Management PAI) |

### 3.3.6 Screen Number

Screens are used to interact with dialog programs and can contain graphical elements such as input fields, labels, pushbuttons, frames, radio buttons, table controls, etc. A four-digit number uniquely identifies each screen in a dialog program.

|  |  |  |
| --- | --- | --- |
| **Format** | **9** | **XXX** |
|  | **9 –** User defined object (fixed)  **XXX –** Sequence number (000 – 999) | |
| **Ex. 9000** (Default screen) | |

### 3.3.7 GUI Status

The GUI Status describes the following elements of a graphical user interface (which are independent from the actual screen layout):

- a menu bar with menus

- a standard toolbar

- an application toolbar

- Functions and function key settings

A single GUI status is generally used to provide users with a consistent graphical interface. Once a GUI status is set it remains active and applies to all screens until a new GUI status is set. Where screens may have fewer active functions, you can use the same status and deactivate one or more of its functions.

|  |  |
| --- | --- |
| **Format** | **<desc>** |
|  | **Ex. CUSTUPD1** (Main Customer Update GUI Status) |

### GUI Title

The GUI Title describes the title of a screen. Once a GUI Title is set it remains active and applies to all screens until a new GUI Title is set.

|  |  |
| --- | --- |
| **Format** | **<desc>** |
|  | **Ex. INQ** (Inquiry GUI Title) |

### 

### 3.3.9 Transaction code

A transaction code executes an ABAP executable program / Dialog screen / OO Transaction or transaction with variant.

Customized programs should be assigned a transaction code or assign it to a menu tree using SE43

(Area Menu).

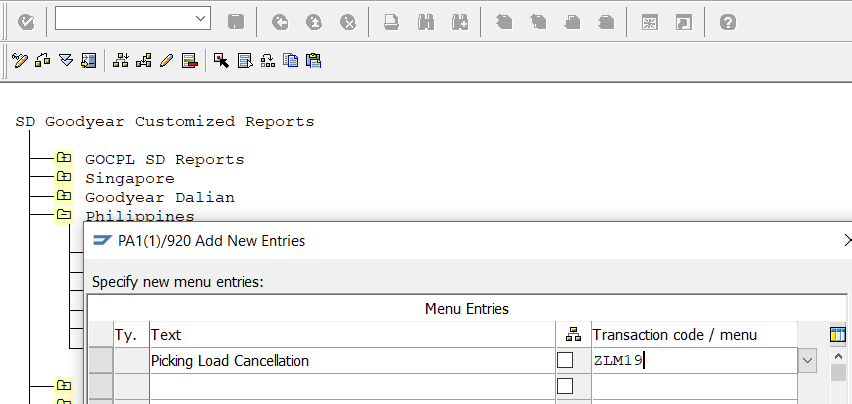
*3.3.10 Using SE43 (Area Menu)*

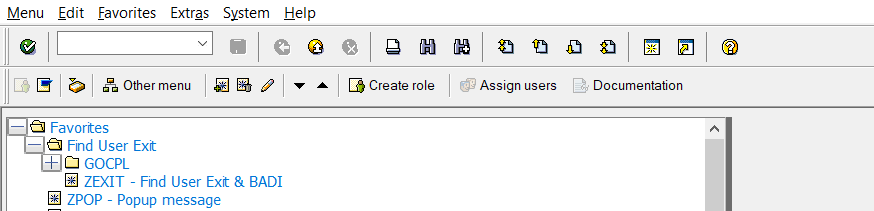
A transaction code will be automatically generated when you assign customized program in the menu.

It is preferred that we do not use this method of program assignment to a menu tree.

It is preferred that the program to be assigned to transaction code using SE93, then assign the transaction to the menu below.

(see *3.3.11 Using SE93 (Maintain Transaction Codes)* below on transaction code naming convention)





*3.3.11 Using SE93 (Maintain Transaction Codes)*

For manually assign transaction code, use the following format.

|  |  |  |  |
| --- | --- | --- | --- |
| **Format** | **Z** | **YYY** | **####** |
|  | **Z** – Namespace (fixed) | | |
| **YY** – Module (see table 2.2) | | |
| **\_** – Separator | | |
| **####** - Numeric (Follow sequence using TSTC table for number range 0000-9999) | | |
| **Ex. ZSD0001, ZMDG0001** | | |

## *DICTIONARY OBJECTS*

### Tables

A table is a two-dimensional matrix that describes a relationship in the database system. A table is identified uniquely by one or several fields. It is recommended that all tables be defined as transparent tables. Tables are maintained via transaction SM31.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **T** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **T** – Object type – T (fixed, see table 2.1) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZFIT\_FRECONC** (Financial Accounting Table) | | | | |

### Structures

A structure is similar to a table. Unlike a table, no data is stored in a structure. It is used in particular for defining data at the interface between module pools and screens, and for standardizing parameters for reusable FORMs and function modules.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **S** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **S** – Object type – S (fixed, see table 2.1) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZCOS\_COSCENTRE** (Cost Centre Accounting Structure) | | | | |

### Table Types

Table types are construction blueprints for internal tables that are stored in ABAP Dictionary. When you create a table type in ABAP Dictionary, you specify the row type, access type, and key. The row type can be any data type from ABAP Dictionary, that is, a data element, a structure, a table type, or the type of a database table. You can also enter a predefined dictionary type directly as the row type, in the same way that you can with a domain.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **TT** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **TT** – Object type – TT (fixed, see table 2.1) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZCOTT\_COSCENTRE\_ALV** (Cost Centre Accounting Structure for ALV) | | | | |

### Views

A view is a virtual table, tailored to the needs of an application. It is not actually physically stored but is derived from one or more physical tables.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **V** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **V** – Object type – V (fixed, see table 2.1) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZMMV\_LOTSZE** (Materials Management View) | | | | |

### Table Indices

An index is used to speed up the accessing of data from a database table. Indexes are single fields or groups of fields that identify each entry in the table. The table’s key is always the primary index and is automatically defined as index ‘0’ when the table is created. Additionally, secondary indexes can be defined which provide alternate sequencing of the table. This allows the multiple means of accessing table entries, depending on the applications requirements. Table indexes are specific to the table against they are defined.

|  |  |  |  |
| --- | --- | --- | --- |
| **Format** | **Z** | **XX** |  |
|  | **Z** – Namespace (fixed) | | |
| **XX** – Numeric (00 – 99) | | |
| **Ex. Z02** (Secondary Index 2) | | |

### Table Fields

A field is a variable, which cannot be decomposed into smaller parts. Fields are defined by such characteristics as length and type (character, date, integer, etc.). A field is the specific unit of data within a table.

|  |  |
| --- | --- |
| **Format** | **<desc>** |
|  | **Ex. TAX\_CODE** (Sales Tax Code Field) |

### Data Elements

A data element defines the description for the field. The semantic information (description and field labels) is assigned to the data element along with any on-line documentation.

The data element is the functional name associated with a domain: e.g., one of the data elements associated with domain CHAR30 may be ADDRESS. Whenever ADDRESS is entered as a data element, the characteristics of the domain CHAR30 appear.

Documentation must be maintained for the data element. This documentation appears as help text when fields in an on-line program are tied to the data element.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **X** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **X** – Object type – X (fixed, see table 2.1) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZCOX\_TAXCODE** (Cost Centre Accounting Structure) | | | | |

### Domains

A domain describes the physical format of the data. A single domain can be used as the basis for any number of fields that are identically structured.

Fields referring to the same domain (via the data elements assigned to them) are automatically changed when a change is made to the domain. This ensures consistency between the fields.

Domain names should indicate the type of field and its length, following the patterns shown under *Examples*, below. SAP provides numerous pre-defined domains (e.g. CHAR25, DATE8, DEC14\_4, and NUM08) which should be used where appropriate. New domains should only be created when there is no such domain defined in SAP.

Variations on an existing domain should have alphabetic suffixes (\_A, \_B, etc.). The first time a domain is used, do not use a suffix. If similar domains are used in different ways, give them different names. For example, the same 4-digit domain would not be used for both a five-character postal code and quantity; rather, you might create ZOOD\_NUM4 and ZOOD\_NUM4\_A.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **D** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **D** – Object type – D (fixed, see table 2.1) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZSDD\_PLANT** (Plant) **ZOOD\_NUMC10** (NUMC length 10 generic) | | | | |

### Lock Objects

A lock object creates function modules that control simultaneous access to the same data by two users. They are called and released through call function modules, which are automatically generated when a lock object is defined. Whole tables or single records can be locked.

|  |  |  |  |
| --- | --- | --- | --- |
| **Format** | **E** | **Z** | **<desc>** |
|  | **E** – Object type – E (fixed) | | |
| **Z** – Namespace (fixed) | | |
| **<desc>** - Free text (description) | | |
| **Ex. EZXCUST** (Lock for Customer Cross Ref Table) | | |

### Search Help

Search helps are objects that you can use to assign for input help (F4 Help) to screen fields.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **SH** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **SH** – Object type – SH (fixed, see table 2.1) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZSDSH\_PLANT** (Plant Search) | | | | |

### Type Groups

It is possible to define non-elementary (or user-defined) types and constants in type groups. This allows the types or constants to be re-used in other programs. Any changes made to the definitions in the type groups will automatically be reflected in all programs that reference them.

|  |  |  |  |
| --- | --- | --- | --- |
| **Format** | **Z** | **N** | **<desc>** |
|  | **Z** – Namespace (fixed) | | |
| **N** – Object type – N (fixed, see table 2.1) | | |
| **<desc>** - Free text (description) | | |
| **Ex. ZNFI1** (FI Module Type group) | | |

### ABAP on HANA objects

### CDS View

A CDS view serves to define the structure of an SQL view and represents a projection onto one or several Dictionary tables or Dictionary views.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **C** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **C** – Object type – C (fixed, see table 2.1) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZFOSC\_TRUCK** (CDS for Truck) | | | | |

### CDS Table Function

ABAP CDS table functions define table functions that are implemented natively on the database and can be called in CDS. As such, they support the HANA platform code pushdown capabilities in ABAP CDS.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **E** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **E** – Object type – E (fixed, see table 2.1) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZFOSE\_TRUCK** (CDS Table Function for Truck) | | | | |

### ABAP Managed Database Procedure (AMDP)

ABAP Managed Database Procedures are a new feature in AS ABAP allowing developers to write database procedures directly in ABAP. You can think of a Database Procedure as a function stored and executed in the database.

### Restrictions for ABAP/4 Dictionary objects

This is a summary of the restrictions applying to the various objects.

**Table key**

- Must be located together at the beginning of the table.

- May contain up to 16 key fields.

**Length of key**

- Maximum length of the key is 251 - (number of key fields).

- If the key length > 120, certain restrictions apply to the transport of table entries.

When transporting, you can only specify the key up 120

(restriction imposed by the transport system). If the length of the key exceeds 120,

generic transport is necessary.

**Table length**

- Maximum length of a table is 1900 - (number of table fields). An exception to this is tables containing a long field (type LCHR or LRAW). The long field is not included in the length calculation.

**Table fields**

- A table may contain a maximum of one long field (type LCHR or LRAW).

- The long field must be located at the end of the field list. A length field of type INT2 must directly precede the long field.

**Buffering**

- If the key is > 32, the table cannot be generically buffered.

- If the key is > 120, the table cannot be buffered.

**Logging**

- If the key is > 86 or the data section is > 500, the table cannot be logged.

**Indexes**

- A maximum of 16 indexes per table is allowed.

**Views**

- Maximum length of view 4096 - (number of view fields).

**Conversion**

- Pooled or cluster tables with more than 250 fields cannot be converted.

**Lock argument**

- The maximum length of the Lock argument is 120 places.

## *FUNCTION GROUP OBJECTS*

### Function Group

Function modules are general-purpose library routines that are available system-wide. Function groups are used to combine function modules that logically belong together. In other words, every function module belongs to a function group.

When an ABAP/4 program contains a CALL FUNCTION statement, the system loads the entire function group in with the program code at runtime. It is recommended that the function module name is very descriptive.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **G or GM** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **G/GM** – Object type – G/GM (fixed, see table 2.1) | | | | |
| **\_** – Separator (Optional) | | | | |
| **<desc>** - Free text (description, optional) | | | | |
| **Ex. ZMMG** (Function Group for MM objects)  **ZMMGM** (Function Group for MM table Maintenance objects) | | | | |

### Function Modules

A function module is a general-purpose library routine, stored in a central library that is available system-wide. Function modules can be called from within an ABAP program and use a defined interface to pass parameters between programs.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **\_** | **YY** | **F** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | | |
| **\_** – Separator | | | | | |
| **YY** – Module (see table 2.2) | | | | | |
| **F** – Object type – F (fixed, see table 2.1) | | | | | |
| **\_** – Separator | | | | | |
| **<desc>** - Free text (description) | | | | | |
| **Ex. Z\_MMF\_VAL\_DATE** (Validate date) | | | | | |

## *ABAP Objects*

### Class

The type of an object is known as its class. A class is an abstract representation of an object. Alternatively, it can be viewed as a set of instructions for building an object. The attributes of objects are defined by the components of the class, which describe the state and behavior of objects.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **CL** | **YY** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **CL** – Object type – CL (fixed, see table 2.1) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZCLQM\_MATCHAR** (Class for material characteristics) | | | | |
| **\*There is a separate naming for BADI implementing class** | | | | |

### Interface

Interfaces are independent structures that enable the class-specific public points of contact to be enhanced by implementing them in classes. Different classes that implement the same interface can all be addressed in the same way. Alongside inheritance, interfaces provide one of the pillars of polymorphism, since they allow a single method within an interface to behave differently in different classes. Interface reference variables allow users to address different classes in the same manner. Interfaces can also be nested.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **IF** | **YY** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **IF** – Object type – IF (fixed, see table 2.1) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZIFOO\_VIEWS** (Interface for views) | | | | |

### Exception Class

Class based object for exception handling.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **CX** | **YY** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **CX** – Object type – CL (fixed, see table 2.1) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Free text (description) | | | | |
| **Ex. ZCXOO\_VIEWS** (Exception class for views) | | | | |

## *OTHER OBJECTS*

### Messages in SAP Programs

Message IDs are used to add new error messages to an SAP application area. They consist of a two-character message class and a three-digit sequence number. Message text should be as generic as possible to facilitate reuse. Whenever possible, use *&* in the body of the message to represent variables passed from the program (e.g., ‘Customer number & is invalid’).

No changes should be done in SAP Standard message. If a new message is needed due to modification associated with standard code, a custom message class should be created.



### Development Classes / Package

Development classes are used to group together logically related development objects that must be corrected and transported as a group.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | |
| **YY** – Module (see table 2.2) | | | |
| **\_** – Separator (Optional) | | | |
| **<desc>** - Free text (description, optional) | | | |
| **Ex. ZFI** (Package for Global FI objects) | | | |

### SET/GET Parameters

The SET and GET Parameters allow default values to be defined for screen fields. The SET parameter stores the user input in memory and the GET parameter retrieves the previous field content from memory. Activating both parameters results in an entered value being the default value the next time the screen is displayed

|  |  |  |  |
| --- | --- | --- | --- |
| **Format** | **Z** | **K** | **<desc>** |
|  | **Z** – Namespace (fixed) | | |
| **K** – Object type – K (fixed, see table 2.1) | | |
| **<desc>** – Alpha Numeric | | |
| **Ex. ZKM** | | |

### Business Addins

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **BD** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | |
| **BD** – Object type – BD (fixed, see table 2.1) | | | | |
| **YY** – Module (see table 2.2) | | | | |
| **\_** – Separator | | | | |
| **<desc>** - Badi definition name | | | | |
| **Ex. ZMMBD\_MEPO\_CUST** (Badi Implementation for ME\_PROCESS\_PO\_CUST) | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Format** | **Z** | **CL** | **YY** | **\_** | **IM** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | | | | |
| **CL** – Object type – CL (fixed, see table 2.1) | | | | | | |
| **YY** – Module (see table 2.2) | | | | | | |
| **\_** – Separator | | | | | | |
| **IM** – Object type – IM (fixed, see table 2.1) | | | | | | |
| **<desc>** - Free text (description) | | | | | | |
| **Ex. ZCLMM\_IM\_ MEPO\_CUST** (Implementation Class for ME\_PROCESS\_PO\_CUST) | | | | | | |

### Enhancements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | |
| **YY** – Module (see table 2.2) | | | |
| **\_** – Separator | | | |
| **<desc>** - Free text (description) | | | |
| **Ex. ZMM\_ENHANCE** | | | |

Enhancement Implementations / Composite Enhancement Implementations / Enhancement Spots

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | |
| **YY** – Module (see table 2.2) | | | |
| **\_** – Separator | | | |
| **<desc>** - Free text (description) | | | |
| **Ex. ZMMEI\_ENHANCE** | | | |

## *Naming Convention Summary*

|  |  |  |
| --- | --- | --- |
| **Object type** | **Naming Convention** | **Remarks** |
| Package | ZYY\_<name> | YY = Module  \_<name> = Optional |
| Transaction code | ZYY\_<name> | YY = Module  Please use this standard for t-code. |
| Table | ZYYT\_<name> | YY = Module  T = Fixed |
| Index | Z## | ## = Number (00 – 99) |
| View | ZYYV\_<name> | YY = Module  V = Fixed |
| Domain | ZYYD\_<name> | YY = Module  D = Fixed |
| Data element | ZYYX\_<name> | YY = Module  X = Fixed |
| Search help/Match code | ZYYSH\_<name> | YY = Module  SH = Fixed |
| Lock object | EZ<name> | EZ = Fixed <name> should be the name of the table. |
| Message class | ZYY\_<name> | YY = Module  \_<name> = Optional |
| Program | ZYY\_<name> | YY = Module |
| Function group | ZYYG\_<name> | YY = Module  G = Fixed \_<name> = Optional |
| Function group for table maintenance screens | ZYYGM\_<name> | YY = Module  GM = Fixed \_<name> = Optional |
| Function module | ZYYF\_<name> | YY = Module  F = Fixed |
| Class | ZCLYY\_<name> | YY = Module  CL = Fixed |
| Interface | ZIFYY\_<name> | YY = Module  IF = Fixed |
| Exception Class | ZCXYY\_<name> | YY = Module  CX = Fixed |
| BAdI Implementation | ZYYBD\_<name> | YY = Module  BD = Fixed |
| BAdI Implementation – class | ZCLYY\_IM\_<name> | YY = Module  CL = Fixed IM = Fixed |
| Structure | ZYYS\_<name> | YY = Module  S = Fixed |
| Table types | ZYYTT\_<name> | YY = Module  TT = Fixed |

# 4. CODING STANDARD CONVENTION

**All change documentation in the source code must be in English.**

The purpose of this section is to define the basic standards that all developed programs must adhere to this documentation. The standards are given three main headings, formatting standards, general standards and finally coding standards. A skeleton program has been provided in Appendix A.

## 4.1 FORMATTING STANDARDS

### 4.1.1 Program Structures

The ABAP/4 language is an event-driven programming language. The execution of an event is controlled by the ABAP/4 processor. Since the language incorporates many 'event' keywords and these keywords need not be in any specific order in the code, the following program skeleton has been developed to provide commonality in the creation of ABAP/4 programs.

This program skeleton is an example of one style of the sequence of program structures that could be used for application development within the R/3 system.

Use the following example when creating a new program.



### 4.1.2 Placement of Process Blocks

For purposes of clarity, **use top-down structured programming**.

Code should be placed in order of execution wherever possible.

Align the SQL codes:

    TRY.  
      lo\_email\_req = cl\_bcs=>create\_persistent( ).  
\*     get recipients  
      SELECT email FROM zwmlogmail  
        INTO TABLE @DATA(lt\_email).  
      IF sy-subrc EQ 0.  
        LOOP AT lt\_email INTO DATA(ls\_email).  
          TRY.  
          lo\_email\_req->add\_recipient(  
            EXPORTING  
              i\_recipient     = cl\_cam\_address\_bcs=>create\_internet\_address(  
                                    i\_address\_string = ls\_email-email ) ).  
          CATCH cx\_send\_req\_bcs.  
            MESSAGE 'Error creating Email!' TYPE 'E'.  
          ENDTRY.  
        ENDLOOP.  
      ELSE.  
        MESSAGE 'No email recepients in table ZWMLOGMAIL' TYPE 'E'.  
      ENDIF.

### 4.1.4 Program Documentation

Programs should be documented using the SAP documentation object component accessible through the ABAP/4 Editor (SE38).

In addition, **internal source codes comments** are a major source of program and its functions.

**Every program should use internal comments extensively.**

The header portion of your ABAP/4 program should contain the author, date and purpose of your program.

Comments within the code should NOT just be a restatement of the code (Example code below); they should **explain what something is or why something is being done**.

Example:

\*  “Check function code  CASE r\_ucomm.  
    WHEN ‘&IC1’.  
\*  “Check field clicked on within ALVgrid report  
      IF rs\_selfield-fieldname = ‘EBELN’.  
\*  “Read data table, using index of row user clicked on  
        READ TABLE it\_ekko INTO wa\_ekko INDEX rs\_selfield-tabindex.  
\*  “Set parameter ID for transaction screen field  
        SET PARAMETER ID ‘BES’ FIELD wa\_ekko-ebeln.  
\*  “Execute transaction ME23N,

\* “and skip initial data entry screen  
        CALL TRANSACTION ‘ME23N’ AND SKIP FIRST SCREEN.  
      ENDIF.  
  ENDCASE.

.

If possible, any ambiguous lines of code should be commented on the same line using the ABAP/4 double quote “ character.

Example:

  CONCATENATE sy-datum+6(2) ‘.’  
              sy-datum+4(2) ‘.’  
              sy-datum(4)

INTO wa\_header-info.**“**todays date

Example:

\*                   “select Scheduling Agreement Schedule Lines  
      SELECT ebeln ebelp wemng glmng menge  
        INTO (eket-ebeln, eket-ebelp, eket-wemng, eket-glmng, eket-menge)  
        FROM eket  
        WHERE ebeln = ekpo-ebeln AND  
              ebelp = ekpo-ebelp AND  
              wemng = 0 AND            “Quantity of goods received  
              glmng > 0.               “Quantity delivered (stock transfer)

\*                   “on the water formula

        w\_avail-menge = w\_avail-menge + ( eket-glmng – eket-wemng ).  
  
        MODIFY t\_avail FROM w\_avail.  
  
      ENDSELECT.

### 4.1.5 Change Documentation

**This is compulsory for each change request to Staging and Production.**

Any change to an ABAP/4 program, whether a copy of an SAP program or a new program,

must be documented in the modification log

The request id used in the third column should enable the changes to be traced back to the change management system. Code changes related to the change history should be identified by a comment at the right hand side of the modified/inserted lines. The comment should consist of the programmer’s initials and the change request number. If code has become redundant or is incorrect do not delete it comment it out. Whenever possible, inline comments should begin around position 45.

Change control within programs will follow the guidelines below:

• All program changes will be documented at the top of your program providing for:

***Date position 5***

***Helpdesk Reference position 30***

***Programmer’s ID position 50***

***Description position 4***

***Change reference position 70***

Initial program creation:

**\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*  
\*\*        Name: ZTEMPLATE.  
\*\*    Function: Goodyear APAC ABAP program template.  
\*\*  
\*\*                                               Transaction:  
\*\*----------------------------------------------------------------------  
\*\*  Program function comments...**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*  Date: 11-AUG-2010        10-12345678         Created by: ZA12810  
\*\* Descr: Initial release  
\*\*----------------------------------------------------------------------**

Subsequent program change:

**\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*  
\*\*        Name: ZTEMPLATE.  
\*\*    Function: Goodyear APAC ABAP program template.  
\*\*  
\*\*                                               Transaction:  
\*\*----------------------------------------------------------------------  
\*\*  Program function comments...  
\*\*  ...  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*  Date: 11-AUG-2010        10-12345678         Created by: ZA12810  
\*\* Descr: Initial release  
\*\*----------------------------------------------------------------------  
\*\*  Date: 01-dec-2010        10-12345679         Modified by: ZA12810  
\*\* Descr: Remove codes                                               "01)  
\*\*----------------------------------------------------------------------  
\*\*  Date: 29-dec-2010        10-12345670         Modified by: ZA12810  
\*\* Descr: Implement new selection                           "02)  
\*\*----------------------------------------------------------------------**

1. In the source listing, identify the changed code with reference to the description at the top of your program by the following reference at position 70

Delete block of codes for 1 line modifcitaion:

*\*<< “01)*

*\* read table gt\_temp with key matnr = gt\_temp2-matnr binary search.*

*\*>> “01)*

OR

*\* read table gt\_temp with key matnr = gt\_temp2-matnr binary search. “01)*

Insert block of codes:

\*<< “02)   
      SELECT matnr werks labst  
        INTO TABLE t\_labste  
        FROM mard  
       WHERE matnr IN s\_matnr  
         AND ( werks LIKE 'U%' OR werks LIKE 'N%' ).  
  
      LOOP AT  t\_labste.  
  
        t\_labste1-matnr = t\_labste-matnr.  
        t\_labste1-werks = t\_labste-werks+0(1).  
        t\_labste1-labst = t\_labste-labst.  
  
        COLLECT t\_labste1.  
  
      ENDLOOP.

      CLEAR t\_labste1.

    ENDIF.  
\*>> “02)

The program should be formatted to ensure readability. A program that is easy to read is easier to support.

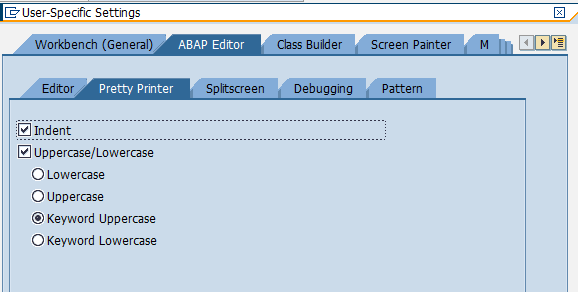
The following is list of points that aid readability:

* Although ABAP allows multiple commands on the same line, place each command on a different line.
* If a continuous section of code exceeds 10-15 lines, create a FORM for the code.
* Minimize the amount of in START-OF-SELECTION. The code in these statements should in most cases consist of PERFORM or CALL METHOD statements.
* Statements that process multiple parameters such as move and write should be formatted with each parameter on a separate line.

## PRETTY PRINTER

### Pretty Printer Configuration

ABAP Developers should use the following settings in their Pretty Printer:



### Pretty Printer Format Execution

Pretty printer should be executed on:

- New custom program

- Modified program where pretty printer in the needed format is already applied

NOTE: Do not use pretty printer for programs that is not formatted pretty printer. Otherwise version management will tag these format changes as modified version of the code.

## CHANGE OF STANDARD CODE CLONES

### Naming Convention

NOTE: SAP Standard modification is highly discouraged and requires approval stating justification before implementing. Only make this action if necessary when there is no available enhancement, BADI or userexit that will be aligned with requirement.

If changes are being made to SAP delivered modules, it is imperative that it should be easy to identify what change was made to the program, when it was made, why it was made and who made it. All modification of SAP standard must be done in the modification mode.

All changes to copied modules, which have been renamed to begin with a ‘Z’, should be identified using the following standard.

### Identifying changes

Each change must be identified with what SAP standard program was copied/cloned:

when the change was made

who made the change

what was changed

why it was changed

a description of how this impacts the user and / or processing:

Before the lines of code that are inserted, deleted or changed, mark the change as follows:

***\*<< “01)***

After the changed, inserted or deleted lines, mark the end of the change as follows:

***\*>> “01)***

* ***Every change in the program should be marked by the change tag specified in the revision history.***

### Change Tag for Deleting Codes

When modifying SAP delivered ABAP/4 programs, existing code should be commented out, not deleted. Put the asterix sign (\*) at the beginning of the line to comment-out the line.

Begin the comment codes as follows:

***\*<< “01)***

***\* Comment out codes…***

End the comment codes as follows:

***\*>> “01)***

\*<<  
\*      clear w\_itab-zzprd2.                                          “01)  
\*      if s688-werks = ‘A119’ or s688-werks = ‘A169’ or s688-werks = ‘A179’.  
\*         w\_itab-zzprd2 = s688-zzprd.  
\*      endif.  
\*>>                                                                  “01)

### Change Tag for Inserting Code

Begin the comment codes as follows:

***\*<< “01)***

***Insert codes…***

End the comment codes as follows:

***\*>> “01)***

\*<<                                                                  “02)  
  CLEAR w\_itab-zzprd2.                                  
  IF ( t001k-bukrs = ‘A110’ OR t001k-bukrs = ‘A120’ ) AND         
         ( s688-werks = ‘A119’ OR s688-werks = ‘A169’ OR

s688-werks = ‘A179’ ).  
        w\_itab-zzprd2 = s688-zzprd.      “Receipts from Production  
  ENDIF.  
  IF ( t001k-bukrs <> ‘A110’ AND t001k-bukrs <> ‘A120’ ).  
        w\_itab-zzprd2 = s688-zzprd.      “Receipts from Production      
  ENDIF

\*>>                                                                  “02)

## 

## CODING STANDARDS

### **Program Heading**

If the ABAP program generates a report, use REPORT <pgmname>, else use PROGRAM <pgmname>.

### **Field and Data Manipulations**

#### Selection-options/Parameters

* It is recommended that selection fields at organizational levels such as *Company code, Sales org, Plant, Controlling area* should be declare at selection-option/parameters.
* Select-options/Parameters of **key** fields that are use in select statement is recommended to be declared as **mandatory**.

This is to prevent selecting all entries from the table unnecessary.

SELECT-OPTIONS: s\_werks FOR ekpo-werks OBLIGATORY,     *"Plant*

s\_bukrs FOR ekko-bukrs OBLIGATORY,      *"Company Code*  
            s\_ebeln FOR ekko-ebeln.

PARAMETERS: p\_budat LIKE bkpf-budat OBLIGATORY

DEFAULT sy-datum.      *“posting date*

#### Data

Although ABAP/4 initializes all fields at program execution, it is always good form to initialize variables using the CLEAR statement. Always clear data structures and variables before repopulating, especially in loops. This highly recommended in function modules since multiple calls to the same function module within one program execution will not refresh the function modules variables. Align data definitions into a readable and consistent format.

#### Variable Typing

Completely qualify all fields.

Instead of: LIFNR

Use: KNA1-LIFNR

#### TYPE

Where possible use the TYPE parameter to declare internal field names. This means that any changes made to the database definition in the data dictionary will be automatically carried forward to the ABAP.

#### CASE

When using CASE, put the most frequently used occurring conditions first.

#### Internal Fields for Constant

Set up frequently used constants as an internal field so that a change to the constant requires a change in only one place.

### **Authorization Checking**

Authorization objects should be checked at the selection screen of the report programs or at the transaction level. Please see notes on section 4.5.

### **Multi-level IF Statements**

Avoid use of multi-level IF statements. If the nesting goes beyond 3 levels use subroutines (FORM), CHECK or CASE statements for clarity.

Use parenthesis to group conditions.

### **Standard Functions**

The use of standard SAP functions such as date routines should be used whenever possible.

### **Return Codes**

All database activity should be followed by a check of the return code SY-SUBRC with appropriate action being coded in the program.

### **Internal Tables**

Do not use REFRESH since the code is obsolete for internal table.

### **Field Symbols**

Assigned/Unassigned checking should be used in a field symbols before using within the program. This will prevent any program terminations within the program.

### **Text Symbols**

Text symbols are to be used whenever text is required in a program, rather than having the text hard-coded. This allows text to be maintained independently of the source code and allows the program to be used with different languages without modification

Instead of: write: / “General Ledger Report for Finance”.

Use: write: / text-001.

OR

Instead of: write: / “General Ledger Report for Finance”.

Use: write: / ‘General Ledger Report for Finance’(001).

### **Subroutines**

Use local variables whenever possible within subroutines for modularization purposes. However, frequently used subroutines within one program call can use global variables to eliminate creation time of those local variables.

It is recommended to pass data to subroutines using the ‘USING’ statement in the PERFORM statement. ‘CHANGING’ clause can be used when passed variable to PERFORM statement needs modification.

Subroutines that are to be called by multiple programs should be created as a function module. External subroutines should also be implemented as Function Modules.

### **CHECK, EXIT, RETURN**

Use these statements to suspend processing and/or skip remaining unnecessary processing for improved performance.

### **AT PFnn**

Use the AT USER COMMAND instead of AT PFnn. This ensures proper response to the user command and it is more readable.

### **EXPORT TO MEMORY**

Do not export entire record to memory if only a few fields are needed. Instead, export by field name.

### **Attributes**

When copying programs, attributes must be changed, if appropriate, to reflect attributes of the new program. For example, the logical database may be different. The program/report name on the first line of the source code will also contain the name of the program copied, so make sure that name is also changed to reflect the new name of the program.

### **User Interface (GUI)**

GUI statuses should be used for interactive report programs and online programs. Use menu bar linking whenever possible to build consistent GUI statuses for screens within a module pool.

### **Native SQL**

Native SQL will not be used to access SAP tables. All access to SAP tables must be done through Open SQL techniques. Native SQL queries do not execute inside ABAP/4 so they circumvent processing for many SAP-specific database features e.g. matchcodes, automatic change documents etc. If Native SQL is used to update tables maintained with these features, inconsistencies will be created.

### **Data Update**

SAP database files are only to be updated through SAP provided code. This can be used through the existing SAP transactions/BAPI or Batch Data (BDC) driven screen processing.

### **Online Help**

In addition to the documentation provided in the program a brief description of the program should be available on-line. In particular any unusual aspects, dependencies etc of the program should be documented. This documentation is provided primarily for support programmers who may not have access to the technical specification for the program.

### **Legacy System Programs**

All legacy system programs (Ex: COBOL/View) that are required as an integral part of batch interfacing to and from the R/3 system must consider the following guidelines when naming the program.

The distinguishing factor in naming the program will be who will take ownership of the program after it is written. If the legacy application area will take ownership of the program then you must name the program according to their standards.

If the R/3 application developers will take ownership of the program, then you must follow the Goodyear application development naming convention for programs. Akron mainframe program series U51 has been reserved for the SAP R/3 Chemical Division project. U52 has been reserved for the SAP R/3 Engineered Products Division project. U53 is also reserved and may be used in the future if necessary.

### **Object Oriented Programming**

The following list is a general guideline when writing Object Oriented Objects.

|  |  |
| --- | --- |
| **NORMAL CLASS** | |
| 1 | Classes are modeled correctly; If a separate class can be limited as an attribute or method of the first one, don't create a new one |
| 2 | Work with global classes as default. Use local classes only where appropriate. |
| 3 | Prefer objects to static classes |
| 4 | Always check the global class repository (SE24) before creating new classes |
| 5 | Make classes that are not explicitly designed for inheritance final. |
| 6 | Prefer returning to exporting. |
| **INHERITANCE** | |
| 7 | Superclasses must have common attributes; If no common attributes can be defined, use interfaces instead for common methods |
| 8 | Always call the super->constructor in your subclass's constructor when the superclass has a constructor of its own |
| **EXCEPTION CLASSES** | |
| 9 | Exception classes are always created globally |
| 10 | Check first if an exception class already exists before creating a new one |
| 11 | All raised exception classes must be caught by the program |
| 12 | Avoid using CX\_ROOT in the CATCH expression; as much as possible, use the immediate exception class |
| **OTHERS** |  |
| 13 | Always use TRY ENDTRY when using widening cast assignments |
| 14 | As much as possible, limit the visibility to private or protected section and use public visibility only on methods that will be called outside the classes |
| 15 | Provide adequate documentation especially when using polymorphism, inheritance, interface, and casting |

## GENERAL STANDARDS



### **Correction and Transport System**

The process of assigning an object to a correction automatically locks the object to other users. This lock will remain in place until the correction is released. All of our production programs will be managed using the Solution Manager CharM.



Programs developed as local private objects cannot be transported. Therefore, programs should only be designated as local private objects if there is no intention of ever using the program in a production environment.

### **Batch Input Program Designs**

Batch input sessions are used to import large volumes of data into the SAP system. In order to avoid performance problems use the general rule of thumb that the BDC sessions should be limited to 250 transactions per BDC group. If it is likely that this limit will be reached, create a smaller number of BDC sessions instead of one large one.

* 1. 1. **Batch Interface BDC Session Names**

The BDC session names consist of 12 characters in length and should take on the format <Program Name><Free Text>.

Note that UPI only allows you to create a 10-character name and then appends a 2-digit sequence number to it.

There may be cases where multiple transactions would be running under one batch session name. This would insure that the second transaction would not start until the first was completed. In situations such as this, simply pick either of the transactions to represent the first four characters of the batch session name.

### **Module Pools**

When generating on-line module pools, the naming conventions, which are used by default, should not be changed.

### **Altering an Active Objects**

If it is planned to alter an active object, SAP recommends that a temporary version of the object be created. If this is done, it is possible to restore the old active object even after the modified version of the object has been activated. When a correction is released, all temporary versions are deleted.

### **Function Groups**

While a single function group would make it easier to find all new functions, it may be more practical to define a number of function groups. This will allow greater flexibility in allowing individual developments or changes to be released and transported without affecting functions being developed or modified by other developers. The number of function groups will be determined by the number of functions to be developed.

### **Maintenance/Correction of Production Codes**

If major coding changes are to occur, indicate the changes in an extended description and revamp the program. The program, however, must comply with the standards defined in this document at the time the modification is to take place and not rely on the old standards when the program was originally developed. This will ensure the reworked program is up to date on new standards and practices.

### **Use Include Modules**

If modifications are made to an SAP delivered program, modifications should be grouped into an include module whenever possible. This will facilitate re-applying changes during release upgrades.

### **Country Specific Codes**

For country-specific development objects, the developer can add '\_XX' at the end of the object name where XX stands for the 2-char country code in table T005L. These string append will be part of the free text.

**4.5 AUTHORIZATION CHECK**

Executable program with selection parameters **is recommended** to implement authority check.

Main check is on the organizational levels.

Eg. company code, sales organization, purchasing organization

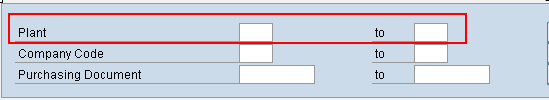
Other levels may be applicable.

Eg. Purchasing Document Type, Sales Document Type

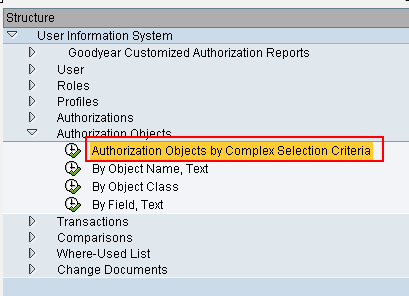
1. From the selection parameters, determine the field to be check.

For this case, Plant.

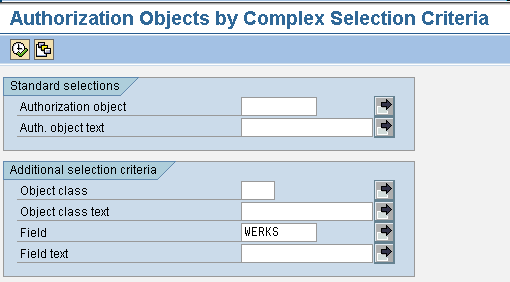
The field name: SELECT-OPTIONS: **s\_werks** FOR ekpo-WERKS.



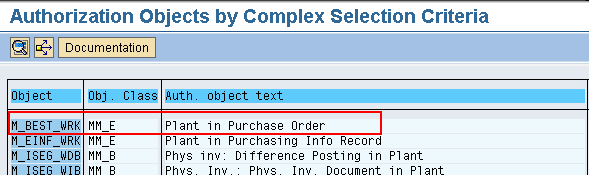
1. From the transaction code SUIM, find the correct authorization object.



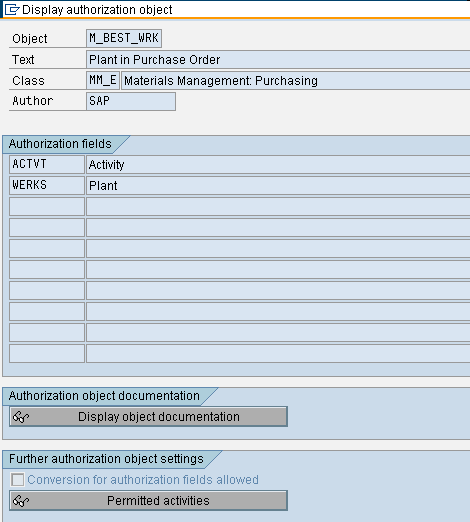
1. Enter the field name, WERKS in ‘Field’ below to find the correct authorization object.



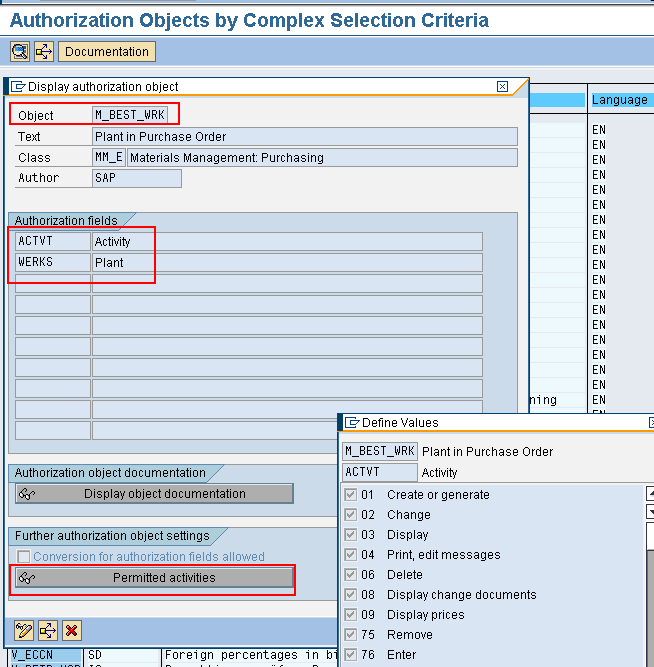
List of objects with field WERKS



Select the object M\_BEST\_WRK



Select ‘Permitted activities’ to list the authorization values.



Base on the objects, build the authorization check:

  AUTHORITY-CHECK OBJECT **'M\_BEST\_WRK'**      
    ID **'WERKS'** FIELD s\_werks  
    ID **'ACTVT'** FIELD '03'.  
  IF sy-subrc <> 0.  
    MESSAGE 'No authorization for plant ' TYPE 'E'.  
  ENDIF.

# 5. MODIFYING SAP PROGRAMS

## 5.1 NAMING CONVENTIONS

**SAP programs must not be directly modified**, since that would not guarantee the integrity of the existing program in future releases. Instead, enhancement techniques should be used or programs should be copied into a new program with the same name and append to the first position of the SAP name by a ‘**Z**’. This will allow the changes to be tracked back to the original program.

**Only standalone programs (i.e. report programs) can be cloned**. Programs called by other programs should not be renamed as all calling programs would have to be modified to call the new program.

Eg. RCPBI010 would be ZRCPBI010

RAMUST00 would be ZRAMUST00

If, however, the SAP program is just being copied and a totally new program is being developed that does not have to be tracked, the name of the new program would fall under the standard naming conventions.

|  |  |  |
| --- | --- | --- |
| **Format** | **Z** | **<desc>** |
| 30 Characters | **Z** – Namespace (fixed) | |
| **<desc>** - Modified copied program | |
| **Ex. ZRAMUST00** | |

# 6. PERFORMANCE GUIDELINES

## 6.1 METHODS OF IMPROVING GENERAL PERFORMANCE

1. Use SQL to minimize the number of database accesses.
2. Process only the minimum number of records from the relevant tables by filtering and validating data at the earliest opportunity using SELECT...WHERE or CHECK
3. Process sets of records rather than single records. (Note: It is better to do SELECT SINGLE if only one row is expected).
4. Avoid using SELECT \* and Select only the required fields from the table unless 70% of the field are required.
5. For selecting single record:
6. Use SELECT SINGLE if all key fields where used in WHERE condition
7. Else use SELECT…. UP TO 1 ROWS
8. Avoid use of SELECT…ENDSELECT
9. When using FOR ALL ENTRIES, make sure that the selection table is not initial before doing the select query. If it is empty the select statement is like a statement without ‘WHERE’ condition.
10. Avoid using nested SELECT statement, SELECT within LOOPs.
11. Create indices that allow you to access the desired data directly.

"Create indices that allow you to access the desired data directly. However, indexes are to be created only when absolutely necessary. Before an index can be created eventually in Production, it has to be tested in an environment close to production like TST01. A SQL Trace before and after the "creation of indexes" must be attached to provide evident that the new index help in performance."

1. Adapt the type of WHERE condition to the type of table in use.
2. Use the IN <internal table> operator to bundle records when reading them.
3. Check your SQL with the database trace.

* Note that the Runtime Analysis function provides numerous examples and explanations of how to develop efficient code. This can be accessed via System --> Utilities --> Runtime Analysis --> Tips & Tricks.
* Use transaction code SE30 (ABAP Runtime Analysis) to check the performance of the program.

# 7. SAP QUERY

**SAP Query Components**

In Goodyear query development, the *SAP Query* comprises of five components are use:

*Queries, InfoSet Query, InfoSets, User Groups* and *Translation/Query*

Choose *Tools -> ABAP Workbench -> Utilities* to call up the components individually.

## 7.1 QUERY AREA

The following two areas are available:

*Standard Area (Client-specific)*

The standard query area is intended primarily for use with ad-hoc queries, that is for queries that are created for a specific one-off purpose and never used again.

Queries in the standard area are client-specific.

There is no connection to the Workbench Organizer.

*Global Area (Cross-client)*

In the global area queries are created not only for use in a single system, but also for transporting to other systems.

The queries in the global area are therefore client-independent.

Query objects created in the global area are registered in the Workbench Organizer. They are created and transported using the normal ***correction and transport procedure***.

Query in Goodyear ECC production system can be maintain directly for user with authorization.

* **It Is recommended that query to be develop in the development instance using *Global Area (Cross-client)*  and transport to staging and production.**

## 7.2 USER GROUPS

|  |  |  |  |
| --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **<desc>** |
|  | **Z** – Namespace (fixed) | | |
| **YY** – Module (see table 2.2) | | |
| **\_** – Separator (optional) | | |
| **<desc>** - Free text (description) | | |
| **Ex. ZMM\_001** (Material Management Group Global) | | |

## INFOSETS

***Table Join***

Table join field link must be specified using key fields as far as possible.

Avoid using **non key fields** as join link.

It is *recommended* that the table join should be **no more than two tables**.

This is to avoid performance issues.

***Field Group***

In Field Group, the ABAP codes in the Extra options must follow the ABAP Program Standard defined in this document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Format** | **Z** | **YY** | **<desc>** |
|  | **Z** – Namespace (fixed) | | |
| **YY** – Module (see table 2.2) | | |
| **\_** – Separator (optional) | | |
| **<desc>** - Free text (description) | | |
| **Ex. ZMM\_001** (Material Management Infoset Global) | | |

## QUERY

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Format** | **Z** | **XX** | **\_** | **<desc>** |
|  | **Z** – Namespace (fixed) | | | |
| **YY** – Module (see table 2.2) | | | |
| **\_** – Separator (optional) | | | |
| **<desc>** - Free text (description) | | | |
| **Ex. ZMMSTOCK** (Stock report query Global) | | | |

## AUTHORIZATIONS IN QUERY

Each query should be assigned to a transaction code.

The transaction code naming convention is defined in this document.

# 8. MISCELLANEOUS

## 8.1 ABAP Program Data – General







**ABAP Program data - General**

|  |  |  |
| --- | --- | --- |
| **Program data type** | **Global data** | **Rule in ATC** |
| Parameters | P\_<name> | PARAMETERS |
| Select-options | S\_<name> | SELECT-OPTIONS |
| Types | T\_<name> | TYPES |
| Table types | TT\_<name> | TYPES |
| Ranges | RT\_<name> | DATA |
| Internal tables | GT\_<name> | DATA |
| Structure | GS\_<name> | DATA |
| Constants | GC\_<name> | CONSTANTS |
| Variables | GV\_<name> | DATA |
| Objects | GO\_<name> | DATA |
| Data Reference | GR\_<name> | DATA |
| Badi Class Reference | GB\_<name> | DATA |
| Exception Class Reference | GX\_<name> | DATA |
| Interface Class Reference | GI\_<name> | DATA |
| Field symbols | <name> | FIELD-SYMBOLS |

|  |  |  |
| --- | --- | --- |
| **Program data type** | **Local data** | **Rule in ATC** |
| Types | T\_<name> | TYPES |
| Table Types | TT\_<name> | TYPES |
| Ranges | RT\_<name> | DATA |
| Internal tables | LT\_<name> | DATA |
| Structure | LS\_<name> | DATA |
| Constants | LC\_<name> | CONSTANTS |
| Variables | LV\_<name> | DATA |
| Objects | LO\_<name> | DATA |
| Data Reference | LR\_<name> | DATA |
| Badi Class Reference | LB\_<name> | DATA |
| Exception Class Reference | LX\_<name> | DATA |
| Interface Class Reference | LI\_<name> | DATA |
| Field symbols | <name> | FIELD-SYMBOLS |

**Classes/Interfaces**

|  |  |  |
| --- | --- | --- |
| **Program data type** | **Local data** | **Rule in ATC** |
| Class | LCL\_<name> | CLASS |
| Exception class | LCX\_<name> | CLASS |
| Interface | LIF\_<name> | INTERFACE |
| Attributes | <name> | CLASS-DATA |
| Events | <name> | EVENTS |
| Methods | <name> | METHODS |

**Parameters for functions modules / methods / WF applications**

|  |  |  |
| --- | --- | --- |
| **Parameter type** | **Naming Convention** | **Remarks** |
| Import | IY\_<name> |  |
| Export | EY\_<name> |  |
| Changing | CY\_<name> |  |
| Returning | RY\_<name> | Methods only |
| Tables |  | Not allowed, Exception needed |
| Exceptions | CX\_<name> | Not checked (ATC) |
| Note: Y – V = Variable, R = Range, T = Internal table, S = Structure, O = Objects | | |

**Parameters for forms (Subroutines)**

|  |  |  |
| --- | --- | --- |
| **Parameter type** | **Naming Convention** | **Remarks** |
| Using | UX\_<name> |  |
| Changing | CX\_<name> |  |
| Tables |  | Not allowed / Obsolete |
| Note: X – V = Variable, R = Range, T = Internal table, S = Structure, O = Objects | | |

## 8.2 Web Dynpro Naming Convention

|  |  |  |
| --- | --- | --- |
| **Type** | **Naming Convention** | **Remarks** |
| Component | ZWDC\_<name> |  |
| Application | ZWDA\_<name> |  |
| WebDynpro Chip | ZWD<name>\_CHIP |  |
| Component Configuration | ZWDC\_<name> |  |
| Application Configuration | ZWDA\_<name> |  |
| FPM Component Scenario Configuration | ZWDC\_<name> |  |
| FPM Application Configuration | ZWDA\_<name> |  |