RAKC ABAP Naming Conventions

Updated Date: 16 July 2024
Version: 1.05

RAK CERAMICS

RAKC ABAP Naming Conventions

Contents

RAKC ABAP Naming Conventions

Document semantics

1.1. Document Properties

Owner and contact information

Document Prepared by:	Ajay Bose Sukumaran
Standards Owner:	Rahul Ghosh
Contact Information:	
Responsible Team:	SAP Solution Delivery Team

1.2. Amendment History

Amendment history for document

Revision No	Revision Date	Author(s)	Comments/Major Changes
1.05	16-July-2024	Ajay Bose Sukumaran	Added naming convention for UI5/FIORI applications

Summary of Changes since last revision

Section / Topic	Short Description of the Change
UI5/FIORI	Added naming convention for UI5/FIORI applications

1.3. Distribution

List recipients for distribution of document

Group	Recipient	Role	Level of Involvement

1.4. Approval

List of persons required for approval of document

Name	Title	Date	Signature
Rahul Ghosh	Senior SAP Solution Architect		

2. INTRODUCTION

2.1. Introduction

The objective of this document is to provide the ABAP naming convention for all RAK Ceramics SAP Development Objects.

2.2. Target Audience

This document is intended for:

- ABAP Developers working for RAK Ceramics Solution Delivery team
- Programmers who have been trained in writing ABAP code
- Quality Assurance team with knowledge of ABAP

3. NAMING CONVENTIONS

SAP has reserved name ranges for customer objects and SAP objects. Using these name ranges ensures that your objects will not be overwritten by SAP objects during the import of a new maintenance level or release upgrade. This section lists the naming conventions for customer-developed objects.

The naming conventions follow SAP suggested customer naming standards. SAP requires that almost all the customer object names (except of module pools and lock objects) start with letter 'Z' or 'Y' or belong to a customer namespace. For the transportable objects we will use 'Z' as the first letter of the customer object names.

- 'Z' is to be used for own created objects
- 'Y' is to be used for non-transportable (local) objects.

Letters representing SAP application modules will be part of object names. The table below (Table XX) gives an abbreviation of the application modules to be used.

Application Module	Abbreviation
Asset Management	AM
Basis Components	BC
Business Warehouse	BW
Controlling	CO
Customer Relationship Management	CRM
Environment Health and Safety	EHS
Enterprise Portal	EP
Financial Accounting	FI
Human Resources	HR
Master Data Management	MDM
Materials Management	MM
Plant Lifecycle Management	PLM
Project Systems	PS
Quality Management	QM
Sales and Distribution	SD
Strategic Enterprise Management	SEM
Solution Manager	SM
Supplier Relationship Management	SRM
Traders and Schedulers Workbench	TSW
Warehouse Management	WM
Exchange Infrastructure	XI

Table 1: Application Modules

3.1. Assignment Objects

3.1.1 Packages

The package is primarily a container for development objects that belong together, in that they share the same system, transport layer, and customer delivery status. Packages are designed to help developers **modularise**, **encapsulate**, and **decouple** units in the R/3 System. They are a further development of the concept of the development class used in earlier R/3 versions (< 4.6C).

The Package Builder (transaction **SE21** or **SPACKAGE**) is used to **migrate** existing development classes to packages, making it a new tool for developing and maintaining development classes.

Format:

ZRAK_ xx _ N
| | ↓ Unique Identifier
| ↓ Required Underscore
↓ Application module
Namespace

Position	Description	Values	Meaning
1-5	Namespace	ZRAK_	Always Z
6 – 8	Application module	See above	Describes the application module (see Table 1: Application Modules).
9	Required Underscore	_	Required Underscore
10 +	Unique Identifier	0-Z	Unique Identifier. Possible Uses: 1) Sequential numeric value, or 2) Project/Release Indicator to distinguish multiple projects or releases

Example: ZRAK_MM_KAAR Package for MM Custom Developments by KAAR Team

3.1.2 Enhancement Project (CMOD)

Format:

Z xx
| ↓
↓ Object Identifier
Identifier

Position	Description	Values	Meaning
1	Custom Identifier	Z	Always Z
2-8	Object Identifier	0-Z	Unique Identifier for the Object

Example : ZPGSE007

3.1.3 Function Groups

Function Group is used to group function modules into functional units.

The naming standards will ensure the custom Function Groups will not conflict with present or future SAP delivered function groups, as well as serve to identify groups of related function modules by functional area.

When a Function Group is created, SAP creates a program with the naming convention SAPL*xxxx*, where *xxxx* is the name of the function group.

Format:

Z **xxx** **_** **N**
↓ ↓ ↓ Unique Identifier
↓ ↓ Required underscore
↓ Application module
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2 – 4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required Underscore	_	Required Underscore
6 +	Unique Identifier	0-Z	Unique Identifier: Unique Identifier to identify the process/ application

Example: **ZFI_ACCT** A function group for Financial Accounting.

3.1.4 Message Classes

Format:

Z **xxx** **_** **N**
↓ ↓ ↓ Unique Identifier
↓ ↓ Required underscore
↓ Application module
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
9	Required underscore	_	Required underscore
8+	Unique Identifier	0-Z	Unique Identifier to identify the process/ application

Example: **ZSD_SWAPP** Message Class for Showroom Application

3.2. Program Objects

3.2.1 Function Modules

When you create function modules in an application module, assign them to the corresponding function group defined for that application module.

Own created Function modules

Format:

Z_ xx _ N
| | ↓ Unique Identifier
| ↓ Required Underscore
↓ Application module
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2	Required underscore	_	Required underscore
3-5	Application module	See above	Describes the application module (see Table 1: Application Modules).
6	Required underscore	_	Required underscore
7 +	Freely definable	0-Z	Freely definable and descriptive text

Example: Function module : Z_HR_GET_PERNR

Own created Function modules with a standard SAP Object as basis

Format:

Z Y _ *
| | ↓ SAP original function module name
| ↓ Required underscore
↓ Identifier
Namespace

Position	Description	Values	Meaning
1-2	Namespace	Z_	Always Z_
3	Custom Identifier	Y	Always Y – OWN created object with a SAP object as basis
4-6	Application module	See above	Describes the application module (see Table 1: Application Modules).
7	Required underscore	_	Required underscore
8+	SAP original function module name	0-Z	SAP original function module name

Add in the header documentation box of the function module, behind the FM name “Copy of <Function Module>”

Example:

You have to change the standard SAP function module GET_BANK_ACCOUNT.

Make a copy of the function module and name it as

Function module : Z_YFI_GET_BANK_ACCOUNT

Function Module Parameters

The function module paramters should conform to the following naming convention:

IMPORTING - I<type>_<parameter_name>
EXPORTING - E<type>_<parameter_name>
CHANGING - C<type>_<parameter_name>
TABLES - T<type>_<parameter_name>

where type is

T - for Tables (Multiline)
S - for Structure
V - for Single value
R - for Reference

Example:

Parameter Type	Tables (Multiline)	Structure	Single Value	Reference (to a Class/Interface)
IMPORTING	IT_SO_NUM	IS_HEADER	IV_ORDER_NO	IR_ALV_HANDLER
EXPORTING	ET_OUTPUT	ES_RETURN	EV_VALUE	ER_OBJECT
CHANGING	CT_ITEMS	CS_DATA	CV_RETURN	CR_HANDLER
TABLES	TT_ITEMS	NA	NA	NA

Note: TABLES parameter is obsolete and should not be used for newly created function modules. Use CHANGING or EXPORTING parameters instead.

3.2.2 Module Pools

Own Created Module Pools

Format:

Z **SAP** **t** **xxx** **_** *****
↓ ↓ ↓ ↓ ↓ ↓
Namespace | Identifier | Module Pool identifier | Application Module | Required UInderscore | Freely definable

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Custom Identifier	SAP	Always SAP
5	Module Identifier	See next cell	Type of Module pool, value **M** = for Screen **D** = for dialog **F** = for subroutine **U** = for update program
6 – 8	Application module	See above	Describes the application module (see Table 1: Application Modules).
9	Required underscore	_	Required underscore
10+	Unique Identifier	0-Z	Freely definable

Example: ZSAPMMM_PURCH A module pool created in material management module

Own Created module Pools with SAP Objects as basis

Format:

Z **SAP** **t** **Y** *****
↓ ↓ ↓ ↓
Namespace | Identifier | Module Identifier | Own Created object with SAP object as basis | Last part of SAP original name after module pool identifier

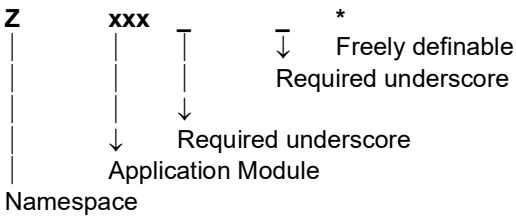
Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Custom Identifier	SAP	Always SAP
5	Module Identifier	See next cell	Type of Module pool, value **M** = for Screen **D** = for dialog **F** = for subroutine **U** = for update program
6	Own created	Y	Always Y
7+	SAP original name	0-Z	Last part of SAP original name after the module pool identifier.

Add in the header documentation box of the Main program, behind the program name “Copy of <Program name>”

Example: ZSAPMYF05V You have to change the standard SAP module pool SAPMF05V

3.2.3 Transaction Codes

Format:



Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6+	Freely definable	0-Z	Freely definable and descriptive text

Example: ZSD_SO_PRINT

3.2.4 Dynpros/Screens (Customer development)

Screens for own created Module pools

xxxx xxxx -> 4-digit numeric 9000 – 9999

Example: Screens for own created module pool

Screens: 9001, 9002……

Screens for own created module pools with SAP Objects as basis

xxxx xxxx -> 4-digit numeric 9900 – 9999

Example: You need to add a screen for a module pool, which is copied from SAP module pool.

Screen : 9900

3.2.5 Dialog Modules

Own created Dialog modules

Format:

Z *****
↓ Freely definable
Namespace

Position	Description	Values	Meaning
1-5	Namespace	Z	Always Z
6-40	Freely definable	0-Z	Freely definable and descriptive text

Own created Dialog module with SAP Objects as basis

Format:

Z **Y** **_** *****
| | | ↓ SAP original object name
| | ↓ Required underscore
| ↓ Identifier
↓
Namespace

Position	Description	Values	Meaning
1-5	Namespace	Z	Always Z
6	Custom Identifier	Y	Always Y – OWN created object with a SAP object as basis
7	Required underscore	_	Required underscore
8+	SAP original object name	0-Z	SAP original object name

3.2.6 Reports/Programs/Enhancements

Own created Programs

Format:

Z xx x _ # _ <ID> _ *

- Z: Namespace
- xx x: Application Module
- _: Required Underscore
- #: Type of Program RICEF
- _: Required underscore
- <ID>: Program ID
- _: Required underscore
- *: Freely definable

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6	Type of Program	See next cell	Type of program, values: **R** = for Reports **I** = for Interfaces **C** = for Conversions **E** = for Enhancements **F** = for Forms **W** = for Workflow.
7	Required underscore	_	Required underscore
8+	Freely definable	0-Z	Freely definable and descriptive text

Example:

ZFI_R_EXCHANGES_RATES	- Report for Exchange Rates in FI
ZMM_C_UPLOAD_CONTRACT	- BDC Program to Upload Contracts in MM
ZSD_E_SHOWROOM_BILLING	- Enhancement for Showroom Sales in SD module
ZHR_W_LEAVE_APPROVAL	- Leave approval Workflow in HR module

Own created Programs with SAP objects as basis

Format:

Z Y Xxx _ *

- Z: Namespace
- Y: Identifier
- Xxx: Application Module
- _: Required underscore
- *: SAP original program name

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2	Custom Identifier	Y	Always Y – OWN created object with an SAP object as basis
3-5	Application module	See above	Describes the application module (see Table 1: Application Modules).
6	Required underscore	_	Required underscore
7+	SAP Original Program name	0-Z	SAP Original Program name

Add in the header documentation box of the program, behind the program name “Copy of <Program name>”

Example: ZYFI_ RFDSLD00 SAP report RFDSLD00 copied and changed

Program Name: Program (head. doc box) : ZYFI_RFDSLD00(Copy of RFDSLD00)

Data Definitions (Global)

	Naming Convention	Example
Type Definitions	TY_<type name>	TY_FINAL, TY_DATA
Global Variables	GV_<variable name>	GV_LINE_COUNT, GV_INDEX
Internal tables	GT_<table name>	GT_UOMS, GT_SALES_INDX
Reference (to a Class/Interface)	GR_<reference name>	GR_PRODUCT, GR_HANDLER
Structures	GS_<structure name>	GS_E1MARAM, GS_HEADER
Constants	GC_<constant name>	GC_BOOLEAN, GC_STATUS_APPROVED
Field Symbols	<FS_<fieldsymbolname>>	<FS_DATA>, <FS_RECORD>

Note: Reduce the usage of global definitions to the maximum possible. Use local declarations and inline declarations possible.

Note: Inline declarations should always follow the local data definition naming conventions mentioned below.

Selection Screen Declarations

	Naming Convention	Example
Parameters	P_<parameter_name>	P_EBELN, P_MATNR
Select-Options	S_<selopt_name>	S_VBELN, S_ORDERS
Check Box	CB_<checkbox_name>	CB_DEFAULT
Radio Button	RB_<radiobutton_name>	RB_TRUE, RB_FALSE
Push Button	PB_<button_name>	PB_SELECT

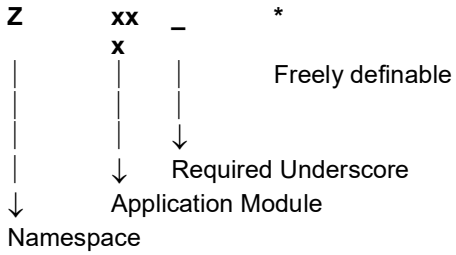
Data Definitions (Locals)

	Naming Convention	Example
Type Definitions	LTY_<type name>	LTY_INTERNAL, LTY_IMP_STRUC
Local Variables	LV_<variable name>	LV_LINE_COUNT, LV_INDEX
Internal tables	LT_<table name>	LT_UOMS, LT_SALES_INDX
Reference (to a Class/Interface)	LR_<reference name>	LR_PRODUCT, LR_BUP_DATA
Structures	LS_<structure name>	LS_E1MARAM, LS_PLANT
Constants	LC_<constant name>	LC_BOOLEAN, LC_STATUS_APP
Field Symbols	<FS_<fieldsymbolname>>	<FS_DATA>, <FS_RECORD>

3.2.7 Includes

Own Created Includes

Format:

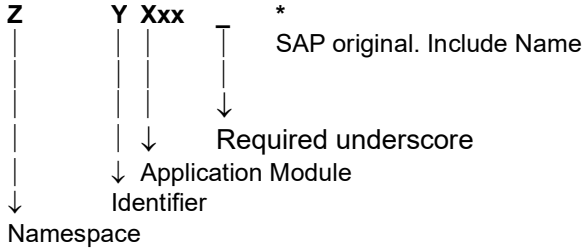


Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6+	Naming should be as explaind in the next column	0-Z	Include name should relate to main program always. For Eg. If main program is ZSD_ORDERS_REPORT then include name should be ZSD_ORDERS_F01 (for subroutines) ZSD_ORDERS_TOP (for data declarations) ZSD_ORDERS_SEL (for selection screen) ZSD_ORDERS_CLS (for class declarations)

Example: ZSD_ORDERS_F01, ZSD_ORDERS_TOP, ZSD_ORDERS_SEL, ZSD_ORDERS_CLS etc

Own Created Include based on a SAP standard Include

Format:



Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2	Custom Identifier	Y	Always Y – OWN created object with SAP include as basis
3-5	Application module	See above	Describes the application module (see Table 1: Application Modules).
6	Required underscore	_	Required underscore
7+	SAP original. Include Name	0-Z	SAP original. Include Name

For example if include FM06LCEK is copied to Z then the name should be ZYMM_ FM06LCEK since the main program RM06EM00 belongs to MM module

Add in the header documentation box of the program, behind the Include program name “Copy of <Include>

3.2.8 Variants

Variants are 14 characters starting with any letter. ‘Z’ or ‘Y’ required for the first letter. Variant names should describe its purpose.

3.2.9 Classes

Classes are templates for objects. Conversely, you can say that the type of an object is the same as its class. A class is an abstract description of an object. You could say that it is 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 behaviour of objects.

Always create classes instead of function modules. Use object oriented programming wherever possible. Create workflow classed instead of Business Objects. For best practices refer SAP help.

Own Created Objects

Format:

Z CL _ xxx _ N

| | | | | ↓ Unique Identifier
| | | | ↓ Required Underscore
| | | ↓ Application module
| | ↓ Required Underscore
| ↓ Class Identifier
↓ Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-3	Class Identifier	CL	Class Identifier
4	Required Underscore	_	Required Underscore
5-7	Application module	See above	Describes the application module (see Table 1: Application Modules).
8	Required Underscore	_	Required Underscore
9+	Freely definable	0-Z	Freely definable and descriptive text

Example: ZCL_SD_SHOWROOM_CART

Own Created Objects with a standard SAP object as basis

Format:

ZY C *
L
| ↓ SAP Original object name
↓ Identifier
Namespace

Position	Description	Values	Meaning
1-2	Namespace	ZY	Always ZY
3-4	Class Identifier	CL	Class Identifier
5+	SAP Original Object name	0-Z	SAP Original Object name

Example: ZYCL_BSP_RUNTIME

3.2.10 Interfaces

Classes, their instances (objects), and access to objects using reference variables form the basics of ABAP Objects. However, it is often necessary for similar classes to provide similar functions that are coded differently in each class but which should provide a uniform point of contact for the user.

ABAP Objects makes this possible by using interfaces. Interfaces are independent structures that you can implement in a class to extend the scope of that class.

Interfaces are extensions to class definitions and provide a uniform point of contact for objects. In contrast to classes, instances cannot be created from interfaces. Instead, classes implement interfaces by implementing all of their methods. You can then address them using either **class references** or **interface references**. Different classes can implement the same interface in different ways by implementing the methods differently. Interfaces thus form the basis for **polymorphism** in ABAP Objects.

Own Created Objects

Format:

Z IF _ Xxx _ N

| | | | | ↓ Unique Identifier
| | | | ↓ Required Underscore
| | | Application module
| | ↓ Required Underscore
| ↓ Interface Identifier
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-3	Interface Identifier	IF	Interface Identifier
4	Required Underscore	_	Required Underscore
5-7	Application module	See above	Describes the application module (see Table 1: Application Modules).
8	Required Underscore	_	Required Underscore
9+	Freely definable	0-Z	Freely definable and descriptive text

Example: ZIF_MDM_PRD_REQUEST

Local Classes (in Programs)

It is recommended to us an “L’ prefix instead of the “Z” prefix when creating Local Classes in custom ABAP Programs. Also as it is local to the ABAP there is no need to project or module identifiers in the naming convention. For example: LCL_PRD_REQUEST.

Own Created Objects with a standard SAP object as basis

Format:

Z Y *
| ↓ SAP Original object name
↓ Identifier
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2	Custom Identifier	Y	Always Y – OWN created object with SAP object as basis
3-4	Interface Identifier	IF	Interface Identifier
5+	SAP Original Object name	0-Z	SAP Original Object name

Example: ZYIF_MDM_PRD_REQUEST

Local Classes (in Programs)

It is recommended to us an "L' prefix instead of the "Z" prefix when creating Local Interfaces in custom ABAP Programs. Also as it is local to the ABAP there is no need to enter project or module identifiers in the naming convention. For example: LIF_PRD_REQUEST.

3.2.11 Methods

Methods describe how an object behaves. You implement them using functions defined within classes. They are operations that change the attributes of a class or interface. There are **instance-** and **static methods**. Instance methods refer to a certain class instance, whereas static methods are common to all the class instances. Static methods can only address static attributes.
Although methods are freely definable it is necessary to have naming conventions for methods and their components so as to use them uniformly within program development. The following conventions are for different types of methods:

Attribute Access

Methods that access attributes of any kind should be prefaced with GET_ or SET_.

Example: GET_STATUS, SET_USE_COUNT

Event Handler

Methods that handle events should begin with ON, followed by the name of the event that they handle.

Example: ON_BUTTON_PUSHED, ON_EVENT_REQUEST

Conversion

Methods that handle data conversions should be prefixed with AS_

Example: AS_STRING, AS_ISOCODE

Boolean Value

Methods that return a Boolean value (true/false) should be prefixed with IS_

Example: IS_ACTIVE, IS_OPEN

Check

Methods that perform check functions should be prefixed with CHECK_

Example: CHECK_AUTHORITY, CHECK_STATUS

The following conventions are for data objects used within the method.

Data Definitions (Within a Method)

	Naming Convention	Example
Type Definitions	LTY_<type name>	LTY_INTERNAL, LTY_IMP_STRUC
Local Variables	LV_<variable name>	LV_LINE_COUNT, LV_INDEX
Internal tables	LT_<table name>	LT_UOMS, LT_SALES_INDX
Reference (to a Class/Interface)	LR_<reference name>	LR_PRODUCT, LR_BUP_DATA
Structures	LS_<structure name>	LS_E1MARAM, LS_PLANT
Constants	LC_<constant name>	LC_BOOLEAN, LC_STATUS_APP

Parameters

The method parameters are regarded from the point of view of the method that implements them and should conform to the following convention:

IMPORTING - I<type>_<parameter_name>
EXPORTING - E<type>_<parameter_name>
CHANGING - C<type>_<parameter_name>
RETURNING - R<type>_<parameter_name>

where type is

T - for Tables (Multiline)
S - for Structure
V - for Single value
R - for Reference

Example:

Parameter Type	Tables (Multiline)	Structure	Single Value	Reference (to a Class/Interface)
IMPORTING	IT_SO_NUM	IS_HEADER	IV_ORDER_NO	IR_ALV_HANDLER
EXPORTING	ET_OUTPUT	ES_RETURN	EV_VALUE	ER_OBJECT
CHANGING	CT_ITEMS	CS_DATA	CV_RETURN	CR_HANDLER
RETURNING	RT_ITEMS	RS_DATA	RV_CODE	RR_HANDLER

3.2.12 Attributes

Attributes contain data. They define the state of an object; that is the instance of a class. Attributes of a class has several attributes themselves, they are:

	Naming Convention	Example
Type Definitions	TY_<type name>	TY_HEADER
Single Value	MV_<variable name>	MV_CART
Internal tables	MT_<table name>	MT_CONFIG
Reference (to a Class/Interface)	MR_<reference name>	MR_PARTNER
Structures	MS_<structure name>	MS_HEADER
Constants	CO_<constant name>	CO_RED

Type

You can specify an attribute as a:

- constant
- instance attribute
- static attribute (that is shared by all instances of the class).

Visibility

You can specify the visibility of attributes for the user of the class. The allowed values are:

- **Public** assigns the attribute to the public area of the class and the attribute can be called by every user of the class. Remember that public attributes form part of the external point of contact to the class, and as such stand in the way of full encapsulation.
- **Protected** attributes are visible and can be used by the class itself and any of its subclasses.
- **Private** attributes are only visible in and available to the class itself. Private attributes therefore are not visible in the subclasses.

Modelled Only

If you have selected this option, the system does not enter the interface in the class pool. You cannot access the components at runtime.

Typing Method

ABAP keyword for defining the type reference. You can choose one of the following: **Type, Like** or **Type Ref To** (for Class/Interface references).

Reference Type

You can use any elementary ABAP type (including generic types) or object type (classes and interfaces).

Read-Only

This option restricts the changeability of attributes as well as visibility. Users cannot change the attribute if a flag is set.

Description

Short description of the components.

3.3. Data Dictionary Objects

Data definitions (metadata) are created and managed in the ABAP Dictionary. The ABAP Dictionary (DDIC) permits a central description of all the data used in the system without redundancies. New or modified information is automatically provided for all the system components. This ensures data integrity, data consistency and data security.

You can create the corresponding objects (tables or views) in the underlying relational database using these data definitions. The ABAP Dictionary therefore describes the logical structure of the objects used in application development and shows how they are mapped to the underlying relational database in tables or views.

The most important object types in the ABAP Dictionary are:

- Tables
- Views
- Types (Table Types, Line Types, Structures, Data Elements)
- Domains
- Search helps
- Lock objects

3.3.1 Tables/Structures/Views/Table Type

Own Created Objects

Format:

Z Xxx _ && _ N

```
|   |   |  |  ↓  Unique Identifier
|   |   |  ↓     Required Underscore
|   |   ↓        Table Type
|   ↓            Required Underscore
|    Application module
Namespace
```

Position	Description	Values	Meaning
1	Namespace	Z	Namespace
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required Underscore	_	Required Underscore
6-			
6-7	DDIC Type	T V S TT LT	Table View Structure Table Type Line Type
8	Required Underscore	_	Required Underscore
9+	Freely definable	0-Z	Freely definable and descriptive text

Example: ZMDM_T_WCC, ZSD_V_SALES, ZMM_S_INVENTORY

Own Created Objects with a standard SAP object as basis

Format:

Z Y *
| ↓ SAP Original object name
↓ Identifier
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2	Custom Identifier	Y	Always Y – OWN created object with SAP object as basis
3+	SAP Original Object name	0-Z	SAP Original Object name

Example: ZYKNA1

3.3.2 Data Elements

SAP created Data Elements should be used whenever available.

Format:

Z *
| Freely definable
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2+	Freely definable	0-Z	Freely definable and descriptive text

3.3.3 Domains

SAP created Domains should be used whenever available.

Format:

Z *
↓ Freely definable
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2+	Freely definable	0-Z	Freely definable and descriptive text

3.3.4 Search Helps

Elementary Search Helps

Format:

Z **xxx** **_** *****
| | ↓ Freely definable
| ↓ Required underscore
↓ Application Module
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6+	Freely definable	0-Z	Freely definable and descriptive text

Example: ZSD_CUST_ADDRESS elementary search help by customer address.

3.3.5 Lock Objects

Format:

Z **E** *****
| ↓ Freely definable
↓ Required by SAP

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2	Required by SAP	E	Always E – Required by SAP
3+	Freely definable	0-Z	Freely definable and descriptive text

3.4. Business Server Pages (BSP)

From basis release 6.10 of the SAP Web Application Server (WAS) the Web Application Builder is fully integrated into the Object Navigator (SE80). This is an integrated platform for developing Web applications based on the *mySAP.com application server*.

The Web Application Builder allows you to develop a new kind of Web application called **BSP applications**. The core parts of these applications are the **B**usiness **S**erver **P**ages (BSPs) and MIME objects.

You can create BSP applications only with the Web Application Builder but not with external Web development environments. The Web Application Builder allows:

- Creation of BSP applications and their pages (BSPs)

- Editing of the layout of BSPs using HTML and the scripting languages ABAP and JavaScript
- Declaration of page parameters for page-based data storage
- Implementation of event handlers using ABAP
- Definition of the page flow using navigation requests
- Integration of the MIME Repository for storing MIME objects
- Creation and definition of themes for adjusting the layout
- Provision of an interface for using external tools for layout implementation using the WebDAV protocol

3.4.1 BSP Application

Like all Repository objects, a BSP application can be created by copying it. Note that the corresponding MIMEs are not copied to the target object when you copy BSP applications. For each new BSP application, the system creates an identically named directory in the MIME Repository, but it is empty.

If you wish to assign the new BSP application to a valid namespace **`XYZ`** that differs from the standard namespace, you must place the namespace as a prefix in front of the actual name of the BSP application. The complete name must then be written in this form: **`/XYZ/BSPApplication`**

Format:

Z xxx _ N

| | | ↓ Unique Identifier
| | ↓ Required Underscore
| ↓ Application module
↓ Namespace

Position	Description	Values	Meaning
1	Namespace	Z	
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required Underscore	_	Required Underscore
6+	Freely definable	0-Z	Freely definable and descriptive text

Example: ZMDM_PRD_REQUEST

Since Basis release 6.20, a service node with the same name is automatically generated in the HTTP service maintenance transaction (SICF) with the new BSP application. As a rule, no such entry exists for BSP applications that were created in earlier releases. In such cases, you must create this node manually in the service maintenance transaction.

Note: The BSP name can be longer than 15 Characters but the HTTP Service must be 15 Characters or less. So DO NOT make your BSP Application name longer than 15 characters. This is the reason why the Project Identifier has been left of the naming convention for this object.

3.4.2 Controller

SAP Web Application Server 6.20 has implemented the Model View Controller (MVC) design pattern, which is widely used in the user interface programming field and which has proved its worth, as an extension of the previous BSP implementation model. Its controller-based use ensures an even clearer distinction between application logic and presentation logic in BSP applications. You can structure graphical user interfaces clearly and organize them in logical units, even with complex applications.

You need to create a controller to use the MVC design pattern in your BSP application. A controller is the instance of a central controller class. In the BSP-MVC environment, each controller is directly or indirectly derived from the same base class CL_BSP_CONTROLLER2, where the central method is `DO_REQUEST`.

Format:

Z Xx .do

| ↓ Extension
↓ Controller name
Namespace

Position	Description	Values	Meaning
Z	Namespace	Z	Always Z
6+	Controller name	0-Z	Name of the Controller, this is Freely Definable and should be descriptive text that defines the function of the controller
Suffix	Extension	.do	Extension name for the controller

Example: Zprd_change.do, Zbup_create.do

3.4.3 Pages

You can implement the layout for each page (often called BSP) using the script languages ABAP and JavaScript.

Format:

Z Xx .ext

| ↓ Extension
↓ Page name
Namespace

Position	Description	Values	Meaning
1-5	Namespace	Z	Always Z
6+	Page name	0-Z	Name of the Page, this is Freely Definable and should be descriptive text that defines the function of the page
Suffix	Extension	.htm	Extension for the Page.

		.bsp	

Example: req_prd_create.htm, _header.bsp, _cell_with_inputfield.bsp

BSPs can have different attributes: Therefore, when you create a page, we distinguish formally between 3 page types:

Page with flow logic

These are simple pages with event handlers, but without much application logic or visualization elements. It is possible to build a BSP application exclusively out of pages with flow logic and event handlers.

Pages with flow logic usually have the suffix ".htm" at the end of their name as they are stand-alone html pages. Example: default.htm, main.htm

View

Views are used solely for displaying application data. They have page parameters in addition to the layout section. In contrast to the page with flow logic, views have neither event handlers nor automatic page attributes.

A controller class is generally assigned to a view. This class controls the view calls and is responsible for communication with the model.

Views are based on the MVC Programming Model and allow you to clearly separate the application logic from the presentation logic in BSP applications.

Views can have the ".htm" or the ".bsp" extension, ".htm" will be used if the view is a standalone html page (usually called first by the controller). ".bsp" is used when the view is a component of an html page.

Examples: req_prd_create.htm, _header.bsp, _logo.bsp

Page fragment

Page fragments are special parts. You cannot use them as stand-alone pages in an application. They are not executable directly. Instead, you include them into other pages using the **include directive**

Page fragments usually have the ".bsp" Extension, this is because they are not usually html pages in there own right.

Examples: _cell_with_inputfield.bsp, _cell_with_button.bsp

3.4.4 MIME Objects

The MIME Repository is used to store all MIME objects (style sheets, graphics, icons and so on) in the SAP System. MIMEs are created as objects in the SAP database and can be referred to on the pages of BSP applications. MIME objects use the SAP development infrastructure. In particular, changes in the MIME Repository, such as the import of new MIMEs, are written to a transport request.

For each BSP application, the system automatically creates an identically named folder in the MIME Repository. This application directory is designed to store all **application-specific MIMEs**. This special folder cannot be deleted explicitly and remains an integral part of the MIME Repository as long as the BSP application exists.

Format:

xx .ext

↓ Extension
Page name

Position	Description	Values	Meaning
1+	Page name	0-Z	Name of the MIME Object, this is Freely Definable and should be descriptive text that defines what the MIME object is
Suffix	Extension	.0-Z	Extension for the MIME (.jpg, .gif, .js etc)

Example: req_prd_create.htm, _header.bsp, _cell_with_inputfield.bsp

3.5. BSP Extensions

Whenever you wish to define and implement your individual tags for Business Server Pages, you first need a BSP extension as a separate workbench. This object then serves as a container for several BSP elements, as a rule

3.5.1 BSP Elements

You create individual elements for a BSP extension and these are inserted later on as tags in BSP pages. Each BSP element has a handler class assigned to it that implements its specific functions. Also, you can create and declare attributes for each BSP element.
Enter the name of the BSP element , a valid name for the *Element Handler Class*, and a meaningful description for the BSP extension as a *short text*.

You can specify an existing, valid ABAP class as the element handler class. A valid class must support the interface IF_BSP_ELEMENT.

We recommend that you derive this class from the automatically-generated basis class(Z)CLG_<name of BSP extension>_<name of BSP elements>. This basis class already contains a standard implementation of the interface methods and is automatically updated whenever changes are made to the element data.

Example: ZCLG_UI_INPUTFIELD

3.5.2 Element Content

In general, BSP elements have a particular content. This means that the resulting tags consist of a start and an end tag, and between these two there are further (embedded) tags, script elements, or even simply just text. By selecting the option for the *Element Content* in the attribute display of the BSP element, you can define whether content should be embedded in the element or not, and – if so – which content.

Example: <ui:inputField id = "carrier"

```
type     = "String"
value    = "<%=carrier%>"
size     = "3"
design   = "standard" />
```

3.5.3 Attributes for a BSP Element

To create attributes for a BSP element, choose the *Attributes* tab in the element view and, if necessary, switch to change mode.

To create and declare an attribute, make the following specifications:

- Attribute
- Required
- Dynamic value allowed
- Pass by reference
- Kind of typing
- Reference type
- Default value
- Description

Examples: id, type, value, size, design

3.6. SAP UI5/ FIORI Application

Format:

Z xxx _ N

```
|   |   ↓  Unique Identifier
|   ↓      Required Underscore
↓          Application module
Namespace
```

Position	Description	Values	Meaning
1	Namespace	Z	
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required Underscore	_	Required Underscore
6+	Freely definable	0-Z	Freely definable and descriptive text

Example: ZSD_SHOWROOM_APP

3.7. Business Add-Ins

Business add-ins are enhancements to the standard version of the system. They can be inserted into the SAP System to accommodate user requirements too specific to be included in the standard delivery. Since specific industries often require special functions, SAP allows you to predefine these points in your software.

As with customer exits, two different views are available:

- In the definition view, an application programmer defines exit points in a source that allow specific industry sectors, partners, and customers to attach additional coding to standard SAP source code without having to modify the original object.
- in the implementation view, the users of Business Add-Ins can customize the logic they need or use a standard solution if one is available.

Format:

Z xxx _ N

| | | ↓ Unique Identifier
| | ↓ Required Underscore
↓ Application module
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required Underscore	_	Required Underscore
6+	Freely definable	0-Z	Freely definable and descriptive text

Example: ZMDM_PRD_REQUEST

3.8. ALE/IDOC Enhancements

3.7.1 IDoc Segment Type

Format:

Z xxx _ _ n
| | ↓ Descriptive text
Required Underscore

| | ↓ Required underscore
↓ Application Module
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z

2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6+	Freely definable	0-Z	Describes the logical grouping of data represented by the segment

3.7.2 IDoc Segment Defenition

Format :

Z xxx _ _ Desc iii
Version number
SAP Description
Required Underscore

Required underscore
Application Module
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6+	Description	0-Z	SAP Description
< >	Version		version number to be filled in by SAP

3.7.3 Basic IDoc Types

Format :

```
Z       xxx  _      _ Desc    iii
|       |    ↓        |       Version number
↓       ↓    |        SAP Description
Namespace    Application Module
             Required underscore
```

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6-28	Description	0-Z	Test Describing the function of the Idoc
29-30	Revision number	01 – 99	It may be useful to resolve similar names for extensions that conflict.

3.7.4 Extension Types

Format :

```
Z       xxx  _      _ X      Desc    ii
|       |    ↓      | ↓              Sequential Number
↓       ↓    |      ↓                Description
Namespace    Application Module       Extension Idoc Type
             Required underscore      Required Underscore
```

Position	Description	Values	Meaning
1-5	Namespace	Z	Always Z
6-8	Application module	See above	Describes the application module (see Table 1: Application Modules).
9	Required underscore	_	Required underscore
12-	Extension Idoc Type	0-Z	Extension Idoc Type

	Description	0-Z	Please use the letters of the customized IDoc that is being extended
	Sequential Number	01 – 99	. It may be useful to resolve similar names for extensions that conflict.

3.7.5 Message Type

Format :

Z **xxx** **_** **_** **Desc**
| | ↓ ↓ ↓
SAP Description
Required Underscore

| ↓ Required underscore
↓ Application Module
Namespace

Position	Description	Values	Meaning
1-5	Namespace	Z	Always Z
6-8	Application module	See above	Describes the application module (see Table 1: Application Modules).
9	Required underscore	_	Required underscore
12-28	Description	0-Z	Please use the description from the IDoc type that will be used with the Message type

3.7.6 Process Code

Format :

Z **xxx** **_** **_** **Desc**
| | ↓ ↓ ↓
SAP Description
Required Underscore

| ↓ Required underscore
↓ Application Module
Namespace

Position	Description	Values	Meaning
1-5	Namespace	Z	Always Z
6-8	Application module	See above	Describes the application module (see Table 1: Application Modules).
9	Required underscore	_	Required underscore

12-28	Description	0-Z	Please use the description from the IDoc type or the basic IDoc type

3.9. Other Objects

3.8.1 SAPscript Forms

Own Created SAPscript Forms

Format:

Z **xxx** **_** *****
| | ↓ Freely definable
| ↓ Required underscore
↓ Application Module
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6+	Freely definable	0-Z	Freely definable and descriptive text

Example: ZSD_ORDERS_DELETED SAPScript in SD module

Own Created SAPScript Forms with standard SAP object as basis

Format:

Z **Y** **_** *****
| | ↓ SAP original object name
| ↓ Required underscore
↓ Identifier
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2	Custom Identifier	Y	Always Y – OWN created object with a SAP object as basis
3-5	Required underscore	_	Required underscore
6+	SAP original object name	0-Z	SAP original object name

Example: ZY_MEDRUCK Change SAPscript form MEDRUCK – Purchasing document

3.8.2 Smartforms

Format:

Z **xxx** **_** *****
| | ↓ Freely definable
| ↓ Required underscore
↓ Application Module
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6+	Freely definable	0-Z	Freely definable and descriptive text
_SF	Suffix for Smartform	_SF	Add Suffix _SF in the end for Smartform

Example: ZMM_PO_PRINT_SF Purchase Order print in MM module

3.8.3 Adobe Forms

Format:

Z **xxx** **_** *****
| | ↓ Freely definable
| ↓ Required underscore
↓ Application Module
Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6+	Freely definable	0-Z	Freely definable and descriptive text
_AF	Suffix for Adobe Form	_SF	Add Suffix _AF in the end for Adobe Form

Example: ZSD_DEL_NOTE_AF Adobe Form forDelivery note print in SD module

3.8.4 Logical Database

Format:

Z **xxx** **_** *****
| | ↓ Freely definable
| ↓ Required underscore
↓ Application Module

Namespace

Position	Description	Values	Meaning
1	Namespace	Z	Always Z
2-4	Application module	See above	Describes the application module (see Table 1: Application Modules).
5	Required underscore	_	Required underscore
6+	Freely definable	0-Z	Freely definable and descriptive text

3.8.5 BDC Sessions

BDC session names are 12 characters. ‘Z’ or ‘Y’ or Z is required for the first letter. The name should describe the purpose of the session.

3.8.6 Background Jobs

Format:

Z **xxx** **_** **<ID>** **_** *****
Namespace
Application Module
Required underscore
Program ID
Required underscore
Freely definable

Position	Description	Values	Meaning
1-5	Namespace	Z	Always Z
6-8	Application module	See above	Describes the application module (see Table 1: Application Modules).
9	Required underscore	_	Required underscore
10-15	Program ID		If known (e.g. INTF001, REPT001) values: **REPT** = for Reports **INTF** = for Interfaces **CONV** = for Conversions
16	Required underscore	_	Required underscore
17-40	Freely definable	0-Z	Freely definable and descriptive text