

ABAP Programming Standards

ABAP Programming Standards, Guidelines   
 and Naming Conventions

History

Version Status Date

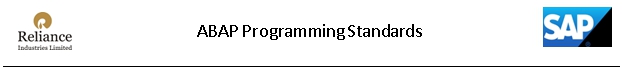
|  |  |  |
| --- | --- | --- |
| **Version** | **Status** | **Date** |
| 0.4 | Modified | Dec-13 |
| 0.3 | Modified | Sep-12 |
| 0.2 | Modified | Jun-12 |
| 0.1 | Initial | Aug-11 |

0.3 Modified September 2012

0.2 Modified June 2012

0.1 Initial August 2011

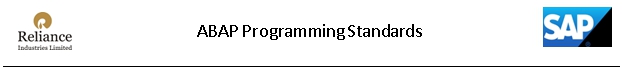
Page: 1/62



Modifications in Naming Conventions (0.4)

|  |  |  |
| --- | --- | --- |
| **Objects** | **Modified (0.3)** | **Modified (0.4)** |
| Global Class Attributes: Constants | lc\_ | lc\_g\_ |
| Global Class Attributes: Work Areas | lw\_ | lw\_g\_ |
| Global Class Attributes: Internal Tables | lt\_ | lt\_g\_ |
| Global Class Attributes: Interfaces | li\_ | li\_g\_ |
| Global Class Attributes: References | lr\_ | lr\_g\_ |
| Global Class Attributes: Objects | lo\_ | lo\_g\_ |

 Page 2/63

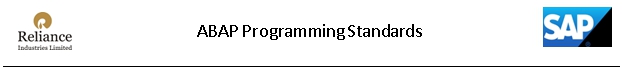


Modifications in Naming Conventions (0.3)

|  |  |  |  |
| --- | --- | --- | --- |
| **Objects** | **Modified (0.2)** | **Modified (0.3)** | **Page No.** |
| Local Variables: Internal Tables | let\_ | lt\_ | 35 |
| ABAP Objects: Global | g\_ | go\_ | 35 |
| ABAP Objects: Local | l\_ | lo\_ | 35 |
| Global Variables: Ranges | gr\_ | Obsolete | 35 |
| Local Variables: Ranges | lr**\_** | Obsolete | 35 |



Page 3/63



Modification in Naming Conventions (0.2)

|  |  |  |  |
| --- | --- | --- | --- |
| **Objects** | **Initial (0.1)** | **Modified (0.2)** | **Page No.** |
| ABAP Programs | ZAAAATXXXXXXXXXXXXXXXXXXXXXXXX | ZXXXXXXXXXXXXXXXXXXXXXXXXX | 36 |
| Module Pools | SAPMZAAAAXXXXXXXXXXXXXXXXXXXXX | SAPMZXXXXXXXXXXXXXXXXXXXX | 36 |
| Includes | M Z AAAA XXXXXXXXXXXXXXXXXXXXXX T NN | M Z XXXXXXXXXXXXXXXXXXXXXX T NN | 37 |
| Transaction Codes | Z AAAA XXXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXXX | 37 |
| Dialog Modules | Z AAAA XXXXXXXXXXXXXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXXXXXXXXXXXXX | 38 |
| Messages | Z AAAA XXXXXXXXXXXXXXXX NNN | Z XXXXXXXXXXXXXXXX NNN | 38 |
| Function Groups | Z AAAA XXXXXXXXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXXXXXXXX | 38 |
| Function Modules | Z\_ AAA\_ XXXXXXXXXXXXXXXXXXXXXXXXX | Z\_ XXXXXXXXXXXXXXXXXXXXXXXXX | 38 |
| Logical Databases | Z AAA XXXXXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXXXXX | 39 |
| Business Add-Ins | Z AAA XXXXXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXXXXX | 39 |
| Classes (defined in DDIC or  programmatically | ZCL\_AAA\_XXXXXXXXXXXXXXXXXXXXXX | ZCL\_XXXXXXXXXXXXXXXXXXXXXX | 40 |
| Tables | Z AAA XXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXX | 42 |
| Views | Z AAA XXXXXXXXXXXXX | Z XXXXXXXXXXXXX | 42 |
| Structures | Z AAA XXXXXXXXXXXXXXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXXXXXXXXXXXXXX | 43 |
| Search Helps (Matchcodes) | Z AAA XXXXXXXXXXXXXXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXXXXXXXXXXXXXX | 43 |
| Elementary Search Help | Z AAA XXXXXXXXXXXXXXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXXXXXXXXXXXXXX | 43 |
| Lock Objects | Z AAA XXXXXXXXXXXXX | Z XXXXXXXXXXXXX | 43 |
| IDOC Segment Types  (Version Independent) | Z AAA XXXXXXXXXXXXXXXXXXXXXXX I | Z AAA XXXXXXXXXXXXXXXXXXXXXXX I | 43 |
| Basic IDOC Types | Z AAA XXXXXXXXXXXXXXXXXXXXXXXX II | Z XXXXXXXXXXXXXXXXXXXXXXXX II | 43 |
| Extension Types | Z AAA XXXXXXXXXXXXXXXXXXXXXXXX II | Z XXXXXXXXXXXXXXXXXXXXXXXX II | 44 |
| Message Types | Z AAA XXXXXXXXXXXXXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXXXXXXXXXXXXX | 44 |
| Process Codes | Z AAA XXXXXXXXXXXXXXXXXXXXXXXXXX | Z XXXXXXXXXXXXXXXXXXXXXXXXXX | 44 |
| Business Object  within the BOR | Z AAA XXXXXX I | Z XXXXXX I | 44 |

Note: In the above list Z can be replaced with Y.





ABAP Programming Standards

Contents

History 1

1 OBJECTIVE 10

2 “GOLDEN RULES” FOR DEVELOPMENT PHASE 11

3 SUMMARY 12

4 CODING STANDARDS 13

4.1 Package and transport request creation 13

4.2 Use of Object Oriented methodology in ABAP developments 13

4.2.1 Why ABAP Objects 13

4.2.2 Usage strategy 13

4.2.3 Obsolete Statements 13

4.3 Object Services 14

4.4 Generic Programming 14

4.5 Use of BREAKPOINT statement 14

4.6 Creation of bespoke table 14

4.6.1 Usage 14

4.6.2 Modification 14

4.7 Report 14

4.7.1 Demo programs and important transactions 15

4.8 Outputs 15

4.8.1 Fixed texts 15

4.9 Enhancements 15

4.9.1 Business Add-ins (BADI) 15

4.9.2 Business Transaction Events (BTE) 15

4.9.3 Customer Exits / Enhancements 16

4.9.4 Form Exits 16

4.9.5 Routines 16

4.9.6 Append Structures 16

4.9.7 Field Exits 17

4.10 Enhancement Framework 17

4.11 The Modification Assistant 17

4.12 Dialog Programming 18

Page: 5/63



ABAP Programming Standards

5 PERFORMANCE STANDARDS 19

5.1 Summary 19

5.1.1 Programming guidelines to achieve scalable programs 19

5.2 Databases Access 20

5.2.1 Use of Appropriate Indexes 20

5.2.2 Keeping the amount of data transferred to the minimum 21

5.2.3 Buffering and Caching 22

5.3 Application Layer 23

5.3.1 Parallel processing 23

5.4 Communication (Dependency on N/W) 27

5.4.1 Minimising round trips 27

5.5 Performance Analysis Tools Usage Metrics 27

5.6 Other performance guidelines 28

5.6.1 Code examples in “Performance Tips and Tricks for ABAP Objects” 28

5.6.2 “Dead” code 28

5.6.3 Subroutine usage 28

5.6.4 IF statements 28

5.6.5 CASE vs. nested IFs 29

5.6.6 Forcing next iteration in a LOOP 29

5.6.7 MOVE-ing structures 29

5.6.8 SELECT and SELECT SINGLE 29

5.6.9 WHERE clause 29

5.6.10 Small internal tables vs. complete internal tables 30

5.6.11 Row-level processing of a table 30

5.6.12 Row-level processing and SELECT SINGLE 30

5.6.13 Reading single records of internal tables 30

5.6.14 Sorting internal tables 31

5.6.15 Number of entries in an internal table 31

5.6.16 Nested SELECTs versus table views/joins 31

5.6.17 If nested SELECTs must be used 31

5.6.18 ORDER BY clause 32

5.6.19 HAVING clause 32

5.6.20 SELECT \* versus SELECT individual fields 32

5.6.21 WHERE clause 33

5.6.22 Avoid unnecessary statements 34

5.6.23 Copying or appending internal tables 34

5.6.24 Declaring internal tables using R/3 Release 4.x syntax 34

6 NAMING CONVENTIONS 36

6.1.1 Transport requests 36

6.1.2 Package 36

6.1.3 Variable naming and declaration 36

6.2 ABAP development 37

6.2.1 ABAP Programs 37

6.2.2 Module Pools & Includes 37

6.2.3 Transaction Codes 38

Page: 6/63



ABAP Programming Standards

6.2.4 Dynpros (Screens) 38

6.2.5 Dialog Modules 39

6.2.6 Messages 39

6.2.7 Function Groups 39

6.2.8 Function Modules 39

6.2.9 Logical Databases 40

6.2.10 Business Add-Ins 40

6.3 ABAP Objects Programming 41

6.3.1 Classes (defined in DDIC or programmatically) 41

6.3.2 Interfaces (defined in DDIC or programmatically) 41

6.3.3 Methods (defined in DDIC or programmatically) 42

6.3.4 Attributes (defined in DDIC or programmatically) 42

6.3.5 Events (defined in DDIC or programmatically) 42

6.3.6 Internal Types (defined in DDIC or programmatically) 42

6.3.7 Constants (defined in DDIC or programmatically) 42

6.4 Data Dictionary 43

6.4.1 Domains 43

6.4.2 Data Elements 43

6.4.3 Tables 43

6.4.4 Append Structures 43

6.4.5 Views 43

6.4.6 Structures 44

6.4.7 Search Helps (Matchcodes) 44

6.4.8 Lock Objects 44

6.5 ALE/IDOC Developments 44

6.5.1 IDOC Segment Types (version-independent) 44

6.5.2 Basic IDOC Types 44

6.5.3 Extension Types 45

6.5.4 Message Types 45

6.5.5 Process Codes 45

6.5.6 Inbound ALE Function Modules 45

6.5.7 Outbound ALE Function Modules 45

6.5.8 Business Object and BAPI developments 45

6.5.9 Business Object Type (within the BOR) 45

6.5.10 Business Object Method 45

6.5.11 BAPI based function module 45

6.6 Workflow developments 45

6.7 Business Server Page (BSP) Applications 49

6.7.1 BSP Application Object 49

6.7.2 Controller 49

6.7.3 BSP page (View, Page with Flow Logic, or Page Fragment) 49

6.7.4 Page attribute 49

6.7.5 MIME object 49

6.8 SAPSCRIPT Development 49

6.8.1 Layout Sets 49

6.8.2 Styles 49

6.8.3 Standard Texts 50

Page: 7/63



ABAP Programming Standards

6.9 Authorizations 50

6.9.1 Authorization Objects 50

6.9.2 Authorization Groups 50

6.9.3 Authorization Object Classes 50

6.10 Legacy System Migration Workbench (LSMW) 50

6.10.1 Projects 50

6.10.2 Sub-projects 50

6.10.3 Object 50

6.11 ABAP Query 51

6.11.1 User Groups 51

6.11.2 InfoSets 51

6.11.3 Queries 51

7 DOCUMENTATION STANDARDS 52

7.1 Header Documentation Box 52

7.2 Comment Box 53

7.3 Comment box before FORM/METHOD declaration 54

7.4 Comment lines 55

8 SPECIAL CONSIDERATION 55

8.1 Internationalisation and Localisation Aspects 55

8.1.1 General Remarks and Concepts in ABAP 55

8.1.2 ABAP Dictionary 55

8.1.3 ABAP/4 Programming 55

8.1.4 SET LOCAL LANGUAGE statement 56

8.1.5 Unicode 57

8.2 Language 60

8.2.1 Text handling 60

8.3 Country specific requirements, 60

9 QUALITY ASSURANCE 62

9.1 Adherence to Programming Standards 62

9.2 ABAP Unit Test 62

10 SECURITY AND AUTHORIZATIONS 63

10.1 General 63

10.2 Reports and Programs 63

Page: 8/63



ABAP Programming Standards

10.3 Viewing Tables 63

10.4 Maintaining Tables 63

10.5 Dialogue, Report and OO Transactions 63

10.6 ABAP Queries 63

Page: 9/63



ABAP Programming Standards

1 Objective

The purpose of this document is to outline coding guidelines, standards and best practices in ABAP The aim of this document is,

To describe a consistent set of practices so that all coding will be done using uniform conventions and techniques. Therefore, once a programmer becomes familiar with these conventions, he will have a much easier time understanding all the other custom developed programs, which also follow the same set of practices.

To recommend, explain and illustrate techniques that have been developed as efficient ways of

handling certain situations in ABAP programming.

To ensure the quality of custom developed programs, without limiting creative development.

Any development work undertaken, as part of the SAP Project will adhere to the guidelines and standards in this document.

Note

This document is not intended to replace SAP ABAP reference material and so it should not be used as means of finding out statement syntax

Page: 10/63



ABAP Programming Standards

2 “Golden Rules” for Development Phase

Know the process and product standards and apply them throughout the whole development   
process.

Check the requirements (for example, given in the solution proposal). Are they clear, complete, and free of constraints?

Write a specification or design. Make clear (to the customer), what he or she will and will not get. Show how the requirements are met by the solution (traceability).

Don’t start realization without sign-off of the specification by the customer and internal review of the   
design.

Stick to your specification or design. As the specification it is approved by the customer (sign-off), deviations must be confirmed again. Any deviations arising thereafter must follow the standard change management process.

Stick to the programming guidelines defined for the project.

Keep your design and coding as simple as possible. This will make the understanding and the followup activities as easy as possible: code review, testing, maintenance, and enhancements.

Careful testing is the best basis to assure quality and customer satisfaction. Remove the cause of a problem and not the symptom.

Think of the impact of your development and inform all affected colleagues.

Page: 11/63



ABAP Programming Standards

3 Summary

Topics covered in this document are listed below, Coding standards

Performance considerations Naming conventions

Documentation

Special considerations   
Quality Assurance

Page: 12/63



ABAP Programming Standards

4 Coding standards

4.1 Package and transport request creation

Packages are designed to help developers modularise, encapsulate, and decouple units in SAP system. In the SAP Project, one package would be created per Track. Any major bespoke development would have a subpackage created. Every technical specification document should have the package information. Developments will not be started without appropriate package.

4.2 Use of Object Oriented methodology in ABAP developments

SAP introduced ABAP objects, the object oriented extension of ABAP as part of release 4.6C. This provides the   
entire required framework for developing programs using Object Oriented methodology.   
They provide all the benefits of Object Orient programming and more,   
 Data Encapsulation

Reusability, through inheritance

Event driven, Applications can be loosely coupled, providing greater flexibility

4.2.1 Why ABAP Objects

Advantages of using ABAP objects and classes,

ABAP objects are explicit, and therefore simpler to use. (Compared to Logical database, reports) ABAP objects offer cleaner syntax and semantic rules. E.g. obsolete and error-prone language   
 concepts are strictly forbidden inside ABAP object classes.

Only way to use new technology SAP Control Framework and BSPs are encapsulated in ABAP object classes.

4.2.2 Usage strategy

For all bespoke developments, first option will be to use Object Oriented programming, unless the OO

programming is an overhead or is not feasible, e.g. like classical dialog programming (using ABAP Dynpro), customer-exits, form exits which involves simple coding requirement.

Following guidelines will be followed,

Use of function modules to minimum, e.g. encapsulating classical screens (Dynpros) or developing   
RFCs.

Use methods instead of subroutines.

Decouple classical screen programming from Application programming using ABAP Objects.

4.2.3 Obsolete Statements

Questionable constructs are still allowed outside ABAP Object classes to support the strict downward

compatibility requirement of ABAP, thereby not requiring changing programs with every new release of ABAP. These statements must not be used in the procedural programs.

Use of these statements is highlighted as warning message in Code inspector and extended program check, and should not be ignored.

List of Obsolete statements in ABAP

1. Go into display mode of any program using transaction SE38. Press HELP Icon or (CNTRL+F8) -  
 > Select ABAP overview radio-button -> press enter -> Select the section Obsolete   
 Statements and Concepts

Page: 13/63



ABAP Programming Standards

4.3 Object Services

Object services provide global services, which ABAP Object language elements cannot provide themselves, to

applications written in ABAP Objects. These Object services include A Persistence Service and Transaction Service. Application objects are managed using Object Service objects know as class actors or agents. There is exactly one class actor for each application class.

4.4 Generic Programming

Special consideration will be given to the use of generic programming techniques to make the code reusable.   
The generic programming concept is based on Object Patterns and uses Run Time Type Services (RTTS) to   
achieve this. Generic programming techniques will be used while developing relevant template programs.

4.5 Use of BREAKPOINT statement

BREAKPOINT statements has been used as part of testing and error analysis. These are static breakpoints and had to be commented before the program is transported to production.

4.6 Creation of bespoke table

The bespoke tables can client dependent or independent, depending on whether the MANDT field is used as   
first key field in the table. It is important to note in case of bespoke table creation, the question to be asked is   
how the data will be maintained, through enhancement /user-exit or manually. In case it is maintained   
manually, then there are two options, creating a dialog program for updating the values or using table   
maintenance generator. Please note that the table maintenance generated transaction have one drawback for   
single screen options, it does not support multi-user maintenance. i.e. only one user can maintain the values at   
any given moment of time. Also activation of buffers will be considered based on the guidelines specified in   
Performance Standards section.

Please note that there will be no generic authorizations for transactions SE16/SE16N and SM30/SM31, so use of authorization group is mandatory for giving access to the right people to maintain a bespoke table. One   
authorization group would be created per Track. All the bespoke tables maintained through table maintenance generator will have an authorization group.

4.6.1 Usage.

The table maintenance dialog option should only be used to set control information, or similar table value that change infrequently, but will change. Examples are tax rates, overhead rates, program control tables, message tables, program or error status tables.

4.6.2 Modification

Also note that there are exits for any enhancement required for table maintenance dialog. Transaction Se55 ->Environment -> Modification

4.7 Report

There are several options for developing report program; classical reports, ALV list, and ALV grid using function   
modules or OO (classes). From WAS6.40 (ESAP will be using this platform) SAP has introduced SAP ALV based   
on a new object oriented programming model supporting ABAP list viewer and SAP control framework   
technology. In all new reporting requirements SAP ALV will be used as a default, refer transaction se83 for   
report template program for The City. You can also find other examples and options in the same transaction.   
Only in the rare case where there is need to copy and change a standard SAP report would options other than   
ALV grid using OO will be used.

Page: 14/63



ABAP Programming Standards

4.7.1 Demo programs and important transactions

Demo programs for SAP ALV - SALV\_LEARN\*, SALV\_DEMO\*, SALV\_FORM\_DEMO\*

For graphical reporting try these program examples GFW\_PROG\* and GFW\_DEMO\_\*

Transactions SE83(Reuse Library) and DWDM(Demo Centre), have demo programs for ALV Grid and use of other controls like tree, text edit.

4.8 Outputs

Outputs can be developed using following tools SAPSCRIPTS, SMARTFORMS and INTERACTIVE FORMS (Adobe). Refer 863893 Interactive forms: Release restrictions (NetWeaver '04)

Preconfigured smartforms supplied by SAP would be used wherever possible. For systems based on Netweaver 04s interactive form (Adobe) should be used as the first option. For WAS6.40 system, Smartforms should be used for developing any new output requirements, SAPSCRIPT will be used only in case there is standard SAP output, which requires a very small change. Though SAP has released Interactive Forms (Adobe) from release WAS6.40, there recommendation is not to use it for productive purpose, instead Smartforms should be used, since it is easier to migrate to Interactive Forms during an upgrade.

4.8.1 Fixed texts

Any fixed text information required in output, will be maintained in standard texts (SO10) and this will be clearly documented in the technical specification document.

4.9 Enhancements

Listed below are enhancement options available and their usage priority i.e. whenever there is an

enhancement required, first check whether this can be done using Easy Enhancement Workbench, if not then check if BADIs can used. If none of the above is available only then other enhancement options should be considered in the order specified in this document.

4.9.1 Business Add-ins (BADI)

For each Business Add-In you have one interface and an adapter class that implements this. The user

implements the interface. BADIs can be multi-instance. In which case more than one implementation can be active at any given moment. Also consider the filter entries, enter ‘\*’ if CLIENT field is one of the filter field.

For any enhancement, BADIs will be preferred over Customer-exits were ever there is an overlap e.g. purchase order enhancement. Please note that with current versions of SAP BADIs also have enhancement for screen, GUI interfaces and tables.

Most of the BADIs can be located through the customising transaction - SPRO or through a search or the statement “CL\_EXITHANDLER” in the program.

4.9.1.1 Transactions

Definition transaction SE18

Implementation transaction SE19

4.9.2 Business Transaction Events (BTE)

BTEs are predecessors of BADI, and available mostly in FI modules. Coding is done in function module which

has fixed interface provide by the SAP. These can be of two types.

Publish and Subscribe in which case the additional activities could be carried out. The data cannot be sent

back to SAP standard.

Process This allows sending back data to SAP standard.

Page: 15/63



ABAP Programming Standards

BTE’s can be located either by using the transaction FIBF or through a search for statement “OPEN\_FI” in the   
program.

4.9.2.1 Transactions

BTE configuration FIBF

4.9.3 Customer Exits / Enhancements

These are one of older enhancements technique available in SAP, and can have function exits, customising   
include, screen exits and menu exits. BADIs should be used instead of customer-exits in case of overlap of   
functionality.

4.9.3.1 Transactions

Definition transaction SMOD

Implementation transaction CMOD

4.9.4 Form Exits

These are enhancements, which use subroutines; these are specific to Sales and Distribution module. Again if there is an overlap of functionality between BADIs and Form exits then BADIs must be used.

A detail documentation of these exits is available in customising - transaction SPRO -> Sales and Distribution -> System Modifications -> IMG activity documentation or you can also find this documentation on SAP help.

4.9.4.1 Notes on Usage

Each form exit or subroutine will have an include program, the coding will be done only in these includes and never in the subroutines.

4.9.5 Routines

These can formulae or requirement routines, which need to be coded, in case SAP standard ones do provide the required functionality. Most of these can be maintained through transaction “VOFM”. Always a copy of SAP standards routine will be made and then changed as per requirement. There is always some amount of customising involved along with the coding in the routine for the solution to work. Also routines would be required in the substitution and validations area in FI module.

4.9.6 Append Structures

Append structures are used for enhancements that are not included in the standard. This includes special developments, country versions and adding customer fields to any tables or structures.

Please ensure the SAP recommended naming convention for naming the append structure fields. The following enhancements can be made to a table or structure with an append structure:   
 Insert new fields

Define foreign keys for fields of table/structure that already exist   
Attach search helps to fields of table/structure that already exist

4.9.6.1 Notes:

You must not use INCLUDE/APPEND structures in BAPI structures, because enhancements to INCLUDE structures generally lead to incompatible changes to the BAPI structure.

Page: 16/63



ABAP Programming Standards

4.9.7 Field Exits

Field Exits are frozen at the maintenance level in 4.6C. No field-exit will be developed on this project.

4.10 Enhancement Framework

With SAP NetWeaver 7.0, the existing enhancement options were significantly increased with the introduction of the Enhancement Framework. Implicit enhancements constituted one of the most important innovations here. You can use them to add your own coding at certain points in programs, such as the beginning of a   
function module or the end of an include. You can also add your own parameters to function modules or   
global methods. The new BAdIs also constitute one of the moist important components of the Enhancement Framework. These are more efficient and flexible than the classic variant.

In ABAP programs, implicit enhancement options are predefined at the following places:

At the end of an include. There are some restrictions, for example, not at the end of a method   
include.

At the end of a PUBLIC-, PROTECTED-, PRIVATE-SECTION of a local class.

At the end of the implementation part of a class (before the ENDCLASS, which belongs to CLASS … IMPLEMENTATION).

At the end of an interface definition (before the ENDINTERFACE).

At the end of a structure definition (before TYPES END OF, DATA END OF, CONSTANTS END OF, and STATICS END OF).

At the beginning and at the end of a procedure (FORM, FUNCTION, METHOD). That is, after

commands FORM, FUNCTION, and METHOD, and before statements ENDFORM, ENDFUNCTION, and ENDMETHOD.

Usually, you should avoid perform-calls completely and use function modules instead. The only

exception to this rule is when you create a form routine instead of a function module in a standard function group in order to be able to access the global memory of the function group.

At the end of the CHANGING-, IMPORTING-, EXPORTING- parameter list of a method in a local class. These enhancement options are located in the middle of a statement.

New variables can be declared within the implicit enhancement options for subroutines, function modules or class methods.

4.11 The Modification Assistant

The Modification Assistant is a tool that makes it easier to reconcile modifications after an update. The

Modification Assistant is integrated into the ABAP Development Workbench and logs all changes that are made to standard objects. It gives you a quick and detailed overview of any modifications that have been made and reduces the amount of time you have to spend on updates.

The Modification Assistant provides support in the following sub areas:

ABAP Editor, Screen Painter, Menu Painter, maintaining text elements, Function Builder, ABAP Dictionary

(enhancing tables and structures with appends, making modifications to the attributes of data elements) and   
documentation (making modifications to documents belonging to particular classes).   
Modifications to objects in the ABAP Development Workbench that are not supported by the Modification   
Assistant are logged and displayed in the overview. Modifications to these objects are reconciled in the usual   
way after updates.

Page: 17/63



ABAP Programming Standards

4.12 Dialog Programming

This development involves creation of screens for capturing manual data entry. With the current version of SAP   
any dialog development should follow the MVC model for GUI development. The development can use any of the   
following techniques ABAP dynpro(SAP GUI), PCUI, BSP or Webdynpro. This depends on the kind of GUI access   
given to the user.

Special care should be taken while programming the database updates, please see the notes on SAP LUW,

logical unit of work and also use of CALL FUNCTION … IN UPDATE TASK in the Performance Standards section. Also using the number ranges and SAP locking could have significant impact on the performance and therefore guidelines mentioned in Performance Standards section should be followed strictly.

Dialog program is always required in case there is need to develop a bespoke application. Question related to data migration should be asked and considered during design and development.

Any ABAP dynpro screen design will follow the SAP guidelines from SAP Style guide.

Page: 18/63



ABAP Programming Standards

5 Performance standards

5.1 Summary

The strength of multi-tier system is scalability, therefore a program can be written on a single server

development system with very few users, but could be deployed on multi server production systems which processes huge volume of data, across continents, with thousands of people using it concurrently. So,   
scalability considerations need to be given when developing any program.

Also in multi-tier architecture, the load optimisation potential is the largest around 40-60% at the application layer. This can be achieved through proper process design, applications and customising.

5.1.1 Programming guidelines to achieve scalable programs Database Layer

Use of appropriate indexes

To optimally use the database, all frequently executed accesses to the database layer are supported by an appropriate index.

Keeping the amount of data read to the minimum.

Buffers and Caches

No identical accesses to database layer

Application Layer

Parallel Processing enabling

Keep Enqueue locking time in programs to the minimum.

Linear dependency

Avoid any memory leaks

There should not be a nonlinear increase in the processing time with increase in amount of data processed.

Communication (Dependency on N/W) Link between presentations Layer Link between servers

Other performance guidelines

Dead code

Subroutine usage IF statements

CASE v/s IF statements Loop statement

Nested SELECT v/s JOIN

Avoid unnecessary statements

SINGLE SELECT is faster than SELECT UP TO 1 ROWS

Page: 19/63



ABAP Programming Standards

5.2 Databases Access

5.2.1 Use of Appropriate Indexes

5.2.1.1 Writing WHERE clauses.

Use the complete key or partial key in the WHERE clause. Ensure that primary key index or appropriate   
secondary index fields are used in the WHERE clause. If a situation arises where there is need to write a

WHERE clause without index, then follow the flow chart below.

Can

WHERE clause YES

NO without index be

avoided?

Can be rewritten Do not use

using proper index NO this statement

YES

Is this statement

rarely used NO

Rewrite with YES

proper index

Can be OK, Adjust the index

This needs to design or

discussed and create a new

documented index

5.2.1.2 Creating Secondary Indexes

A proper index design ensures that the search time is nearly independent of the number of table rows. Columns to be used with EQ (in the WHERE clause) should be at the beginning of index   
 A highly modified table should not have too many indexes

The more columns an index has, the higher the chance that an update will affect one of the indexed   
columns

The cost for maintaining indexes depends heavily on the total length of all index fields

5.2.1.3 Checking proper indexes are being used

5.2.1.3.1 Code inspector - SCI

A static check can be conducted on the program to ensure WHERE clause uses appropriate index on the   
table.

Page: 20/63



ABAP Programming Standards

5.2.1.3.2 Performance Analysis -ST05

Sort column Min.time/r. values higher than 10,000 s, it is quite probable that these database accesses are

not supported by proper index. Use Explain on this statement to check the access path.

5.2.2 Keeping the amount of data transferred to the minimum.

The application should not read data before it is guaranteed to use it. If in doubt, records should be read one   
 by one, as they are needed.

5.2.2.1 WHERE clause

Use WHERE clause, instead of CHECK statement.   
Use

SELECT \* fom   
 WHERE name ENDSELECT.

Instead of

SELECT \* fom

CHECK .

ENDSELECT.

Ensure the data read is actually needed by the application i.e. Accesses to the database layer is with complete WHERE clause.

Note: Use of SELECT FOR ALL ENTRIES

Use DELETE ADJACENT DUPLICATES prior to executing the statement

Check if the internal table were not empty before using SELECT FOR ALL ENTRIES, else all records from the table would be fetched the WHERE clause would be ignored.

5.2.2.2 SELECT \* v/s SELECT field list

Try using projection view in the data dictionary, or use of field list in the select clause.   
Use

SELECT field1 field2   
 INTO

FROM   
WHERE

Instead of

SELECT \*

FROM   
WHERE

5.2.2.3 Use of SAP standard function module

Using the wrong function modules to fulfil a certain subtask can also cause unnecessary database accesses, such as when you call a function that does everything you want plus a few things you don't care about. Each function module used should fit your needs as closely as possible.

5.2.2.4 Make meaningful use of update modules.

Another statement that causes database access without an explicit SQL statement is CALL FUNCTION ... IN

UPDATE TASK. The addition ... IN UPDATE TASK changes a simple function module call, which costs only a few microseconds, into an update call, for which at least two data sets are written to the database. The update   
task reads these data sets to regain interface information so that the function may execute with the correct parameters. After the update is completed, the data sets are deleted again automatically by the system.   
Although using the update task has advantages both for the dialog response time and the locking behaviour of most transactions, you should try to reduce the number of CALL FUNCTION ... IN UPDATE TASK whenever   
possible, which leads to the following guidelines:

Page: 21/63

ABAP Programming Standards



5.2.2.5

5.2.2.5.1



Checking where more data is being fetched than required

Code inspector – SCI

Statements causing unnecessary data reads will be found like, CHECK statements within SELECT and

ENDSELECT, also SELECT statements without WHERE clauses

5.2.2.5.2



Performance Analysis - ST05





Sort the list by column records, which show the total number of records, which have been selected and

transferred by the statement. The higher number than expected is an indication that more entries than

required are read.

Sort the list by column tables. Count the total number of records per database table and compare it with

the number of expected records.

Identify database accesses to the same table entries from different programs. Ensure that the same

information is read only once from the database using appropriate buffer mechanisms

5.2.3

Buffering and Caching

Buffering ensures that the data, which is repeatedly required by programs, can be buffered thus reducing

costly data reads from the database.

5.2.3.1

Avoiding statements that bypass buffers (on tables that are buffered)

Following statements bypass buffers; these statements need to be used with caution. Consider use of

statements that invalidates/bypasses buffer

To take full advantage of buffering, you must be aware of the technical settings of a table as well as the

techniques that bypass buffer processing. The following statements can be used to bypass buffer processing:































SELECT ... BYPASSING BUFFER

SELECT ... FOR UPDATE

Any aggregate function (COUNT, MIN, MAX, SUM, AVG)

SELECT MIN(F1) FROM T1 WHERE ...

SELECT DISTINCT ...

WHERE clause contains ... IS [NOT] NULL

ORDER BY (other than PRIMARY KEY)

GROUP BY or HAVING

JOINs and sub-queries

Use of mandt field in join. Pointer to the Oss note:

SELECT ... FOR ALL ENTRIES - if not all data is buffered

For single-record buffering every statement where key is not fully specified: no SELECT SINGLE

In generic case: no generic key

Any native SQL statement (even INSERT, UPDATE, DELETE).

Comparison between database columns: for example

o

“SELECT \* FROM DBTAB~FIELD1 > DBTAB~FIELD2”

* Check whether interface tables have at least one entry.
* Try to bind together several CALL FUNCTION ... IN UPDATE TASK statements.
* Avoid at all costs calling update function modules within loops.

Page: 22/63



ABAP Programming Standards

5.2.3.2 Buffering tables

Table buffers are best suited for read-only data stored in customizing tables or small tables containing master

data. When you choose a buffer type, make sure that it fits the table size and accesses: It takes much more effort to change a buffered table than a non-buffered one. Therefore, table buffers are best suited for readonly data stored in customizing tables or small tables containing master data.

Note: Buffering allowed but off

In customer system buffering can be switched on(installation-dependent) Specification of buffering type is optional

5.2.3.3 Use of READ modules and SHARED OBJECT Buffers.

Read modules are special function groups(e.g. Using read modules: ME\_EKKO\_SINGLE\_READ) or classes that store information read from the database in global internal tables. The visibility of data is restricted to one SAP logical unit of work (LUW).

Read modules should be used:

When buffering the information using technical setting is not feasible

To prevent unnecessary database calls when the same data is used by different program units of the transaction

To ensure consistent data within one transaction

To check identical selects statements in transaction ST05 Second column “No. of identical selects in percent”.

5.2.3.4 Checking statements which need buffering

5.2.3.4.1 Performance Analysis - ST05

Second column “No. of identical selects in percent”. This statements which are making calls to the

database using same select clause. This is an area where buffering mechanism could be used.

5.3 Application Layer

5.3.1 Parallel processing

Schedule multiple batch jobs

Optimal for long running independent tasks

Use tRFCs

Use for short independent tasks that may run on other systems

Use aRFCs

Useful for short tasks that must be synchronised

SAP Locking and Database Locks

Locking prevents parallel processing. The locking should be for least possible time; unnecessary

locking can cause performance issues. The importance of the lock is determined by how many users are likely to access the object simultaneously. Special attention should be given to the code, which is executed during which the lock is applied. Firstly, on reducing the runtime and also executing at least part of required program steps before lock is applied or after it is released.

Page: 23/63



ABAP Programming Standards

5.3.1.1 Checking Parallel processing bottlenecks

5.3.1.1.1 SAP Enqueue

Monitor the current system wide Enqueue situation with SM66 or SM50

Run Performance Trace (ST05) with option Enqueue Trace on. Display extended trace list to see the

timestamp for each statement. Look for exclusive locks and when they are released either explicitly or implicitly at the end of SAP LUW. Measure the locking time.

5.3.1.1.2 Database Locks

Run Performance Trace (ST05) with option SQL Trace. Display extended trace list to see the timestamp for each statement. Looks for locks set by INSERT, UPDATE, DELETE statements and for next COMMIT WORK, which releases the locks. Measure the locking time.

Make sure that each exclusive lock is necessary and cannot be changed into shared lock.

5.3.1.2 Internal tables

Avoid too large internal tables, Try to keep internal tables small the size of the internal table can be checked in SE30(Runtime Analysis).

Use the right table type:

Standard table for multiple access types, use BINARY SEARCH for mass accesses and try to SORT only   
once

Sorted table, if generic key access is main access type

Hashed table if single line access with fully specified key is only access type

5.3.1.2.1 Hash tables V/S Sorted tables

Hashed tables can only be used if main access uses full table key

Generic accesses are very slow

Best example: Key contains a GUID

Which single line access is faster?

The scaling of hashed-tables is better

However, access costs are similar to sorted tables for sizes up to 100

There is a slight advantage for hashed tables for very large tables Things   
to note:

That internal tables should generally not become too large

Most large tables are large because operations work on generic key ranges and not only on single   
lines

5.3.1.2.2 Sorted Standard tables V/S sorted tables

Sorted Standard Table

Access with binary search as good as access on sorted table Most flexible table

Different access types can be supported Fastest filling method:

If it is not necessary that the table is always sorted during filling Use APPEND , ... , APPEND and one SORT at the end   
 A small number of additions should use INSERT instead of APPEND Sort is expensive as O(n log n).

Every unnecessary extra sort should be avoided

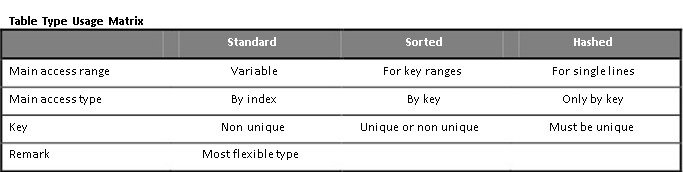
Note: The most expensive SORT is the one on an already sorted table

Sorted tables

Are much more convenient if a specific order must be always guaranteed

Page: 24/63

ABAP Programming Standards



5.3.1.2.3





Filling data into Internal table

From DB tables

Best with array fetch INTO TABLE or APPENDING TABLE

If possible, try to use identical line types as the command INTO CORRESPONDING FIELDS is expensive,

especially for buffered tables

From other internal tables





Best with array fetch APPEND LINES OF or INSERT LINES OF

Beware that MOVE-CORRESPONDING works only on the work area

With COLLECT lines with identical key are aggregated to one line, all numeric fields are summed up

automatically

APPEND

Only for standard table: independent of n

INSERT

Standard table: ~ n

Location (Index) must be determined with READ. Use BINARY SEARCH as insert is only useful on

sorted standard tables. Becomes faster for n > 4096 when a B\* index is used.

Sorted table: ~ log n

Direct INSERT with similar cost as above. Rearrangement is less expensive as a B\* index is used

Hashed table: ~ independent of n

Finding location is cheap. Rearrangement of the hashed function

5.3.1.2.4











Reading data from Internal table

Work area versus header line

Try to avoid header lines, they can cause errors

TRANSPORTING

READ TABLE works with TRANSPORTING field list and ‘NO FIELDS’

LOOP AT ITAB not with TRANSPORTING field list.

Use field list if amount of transported data is reduced considerably

Use TRANPORTING NO FIELDS if only SY-INDEX or SY-SUBRC is needed

ASSIGNING (from SAP R/3 4.5)







Direct table access without copying to work area

Also no MODIFY necessary

Use ASSIGNING especially in nested tables

 Page: 25/63



ABAP Programming Standards

Slowest table access are sequential and grow linearly with n







READ standard table with (table) key

(no BINARY SEARCH)

LOOP at table where

READ hashed table with key (not table key)

Faster access use a sorted index and have a weak dependence on n

Note: n = number of lines in the internal table



READ standard table with key

with BINARY SEARCH (must be sorted)

o Sort statement is expensive; use sort only once.

READ sorted table with table key



Fastest accesses are independent of n

READ hashed table with table key

READ standard table with index

READ sorted table with index

5.3.1.3

5.3.1.3.1

Checking for nonlinearity in processing times

Code inspector – SCI

Poor-performing operations on internal tables select check Low Performance Operations on Internal tables.

Nested loops and selects can be found using following checks,













SELECTs in loops

Changing database access in loops

Nested loops

5.3.1.3.2

Runtime Analysis Trace – SE30

Prepare two test cases. The only difference between two cases is increase of the data size by a factor

of 10(x10).

Run two tests cases and trace them with SE30

Compare entries in the hit list of the two trace files. Runtime with an increase by a factor of more

than 10 indicate highly nonlinear coding.

5.3.1.4

Memory

There can be dynamically growing memory consumption through Internal tables, ABAP / JAVA objects, Strings,

Anonymous data objects. Controlling memory consumption can improve buffer use and let more users work

on one instance.

5.3.1.5

5.3.1.5.1











Checking for nonlinearity in memory usage.

Debugger

Use debugger to create memory snapshot after execution of subsequent tasks/packages/units.

Use transaction S\_MEMORY\_INPSECTOR to compare the memory snapshots

5.3.1.5.2

Tune Summary (ST02)

GOTO -> Current Local data -> SAP memory -> Mode list

Display an overview of the memory size of each user session.

Note the value of the size of the user session after subsequent tasks/packages/units.

Page: 26/63



ABAP Programming Standards

5.4 Communication (Dependency on N/W)

5.4.1 Minimising round trips

5.4.1.1 Communication with presentation server

Network run times include uncertainty factors. Special care needs to be taken while code, which is influenced by Network speed. This can involve communicating with GUI, or with another server. Keep the data exchange to the minimum; follow the guidelines of 2 trips for communication with GUI. Avoid e.g. a program with a greater running time for the SAPGUI Progress Indicator than for the rest of the program, even without the satellite link. A remote function call (RFC) sends this SAPGUI Progress Indicator to the front end. Because a work-process is blocked on the application server during an RFC roundtrip, bad performance at the

presentation-server level influences performance at the application-server level.   
 Make reasonable use of the SAPGUI\_PROGRESS\_INDICATOR

Do not synchronize the contents of controls to often (explicit flush)

5.4.1.2 Server-to-Server communication,

Optimize the number of RFCs

Use RFC connections as long as possible

5.5 Performance Analysis Tools Usage Metrics

Basic Tests SCI ST05 SE30 ST30 ST03N DB OS

Premium Tests monitor monitor

Platform dependent

General Checklist

Appropriate Indexes   
Complete WHERE clauses SAP buffer

No identical selects

X X

X X X

X X X

X X

Parallel processing enabled X

Linear dependency

Two communication steps per dialog steps

X   
 X X X

Actual Measurements

Number of DB calls

Net data volume   
transferred(KB)

Peak memory   
consumption(MB)

CPU consumption of

X X X

X X X

X X X

X X X

Page: 27/63



ABAP Programming Standards

Basic Tests

Premium Tests

Platform dependent

functions / methods (msec)

Number of round trips to presentation server and between servers

Amount of bytes transferred to presentation server

SCI ST05 SE30 ST30

X X

X

ST03N DB OS

monitor monitor

X X

X X

5.6 Other performance guidelines

5.6.1 Code examples in “Performance Tips and Tricks for ABAP Objects”

This contains very good programming tips and tricks for better program performance. You can reach this by getting into any program in display mode using transaction SE38 ->Environment ->Examples->Performance Examples. This has very good examples showing tips on performance improvements. Please use these with due consideration to the assumptions. Some of the examples are covered below.

5.6.2 “Dead” code

Avoid leaving "dead" code in the program. Comment out (or delete) variables that are not referenced and

code that is not executed. Use program --> check --> extended program check to see a list of variables that are not referenced statically.

5.6.3 Subroutine usage

For good modularization, the decision of whether or not to execute a subroutine should be made before the subroutine is called. For example:

This is better

IF f1 NE 0.

PERFORM sub1.   
ENDIF.

FORM sub1.

ENDFORM.

Than this

PERFORM sub1.   
FORM sub1.   
 IF f1 NE 0.

ENDIF.

ENDFORM.

5.6.4 IF statements

When coding IF tests, nest the testing conditions so that the outer conditions are those that are most likely to   
fail. For logical expressions with AND, place the mostly likely false first and for the OR, place the mostly likely   
true first.

Example - nested IF's:

IF (least likely to be true).

IF (less likely to be true).

IF (most likely to be true).

ENDIF.

ENDIF.   
ENDIF.

Page: 28/63



ABAP Programming Standards

Example - IF...ELSEIF...ENDIF:

IF (most likely to be true).

ELSEIF (less likely to be true).

ELSEIF (least likely to be true).   
ENDIF.

Example - AND:

IF (least likely to be true) AND   
 (most likely to be true).   
ENDIF.

Example - OR:

IF (most likely to be true) OR

(least likely to be true).

5.6.5 CASE vs. nested IFs

When testing fields "equal to" something, one can use either the nested IF or the CASE statement. The CASE is   
better for two reasons. It is easier to read and after about five nested IFs the performance of the CASE is more   
efficient.

5.6.6 Forcing next iteration in a LOOP

When the next iteration in a loop has to be executed, use

LOOP AT GT\_ITAB.

CONTINUE.

ENDLOOP.

instead of

LOOP AT GT\_ITAB.

CHECK 1 EQ 2 ENDLOOP.

5.6.7 MOVE-ing structures

When records a and b have the exact same structure, it is more efficient to MOVE a TO b than to MOVECORRESPONDING a TO b.

MOVE BSEG TO \*BSEG.

is better than

MOVE-CORRESPONDING BSEG TO \*BSEG.

5.6.8 SELECT and SELECT SINGLE

When using the SELECT statement, study the key and always provide as much of the left-most part of the key as possible. If the entire key can be qualified, code a SELECT SINGLE not just a SELECT. If you are only   
interested in the first row or there is only one row to be returned, using SELECT SINGLE can increase   
performance by up to three times.

5.6.9 WHERE clause

Although more important in earlier releases of SAP, it is still considered good practice to specify the fields in the WHERE clause in the same order as they appear in the primary key or index of the table.

Page: 29/63



ABAP Programming Standards

5.6.10 Small internal tables vs. complete internal tables

In general it is better to minimize the number of fields declared in an internal table. While it may be

convenient to declare an internal table using the LIKE command, in most cases, programs will not use all fields in the SAP standard table. For example:

Instead of this:

DATA: T\_VBAK LIKE VBAK OCCURS 0 WITH HEADER LINE. USE   
THIS:

DATA: BEGIN OF T\_VBAK OCCURS 0,

VBELN LIKE VBAK-VBELN,

END OF T\_VBAK.

if only a small number of fields from VBAK are actually required in the program.

5.6.11 Row-level processing of a table

Selecting data into an internal table using an array fetch versus a SELECT-ENDSELECT loop will give at least a 2x   
performance improvement. After the data has been put into the internal table, then row-level processing can   
be done.

For example, use:

SELECT ... FROM TABLE <..>

INTO <ITAB> (CORRESPONDING FIELDS OF ITAB)

WHERE ...

LOOP AT <ITAB>

<DO THE ROW-LEVEL PROCESSING HERE> ENDLOOP.

Instead of

SELECT ... FROM TABLE <...

WHERE ...

<ROW-LEVEL PROCESSING> APPEND <ITAB>.

ENDSELECT.

5.6.12 Row-level processing and SELECT SINGLE

Similar to the processing of a SELECT-ENDSELECT loop, when calling multiple SELECT-SINGLE commands on a   
non-buffered table (check Data Dictionary -> Technical Info), you should do the following to improve   
performance:

Use the SELECT into <itab> to buffer the necessary rows in an internal table, then Sort the rows by the key fields, then

Use a READ TABLE WITH KEY ... BINARY SEARCH in place of the SELECT SINGLE command. Note that this only make sense when the table you are buffering is not too large (this decision must be made on a case by case Architecture and Insfrastucture team).

5.6.13 Reading single records of internal tables

When reading a single record in an internal table, the READ TABLE WITH KEY is not a direct READ. This means   
that if the data is not sorted according to the key, the system must sequentially read the table. Therefore, you   
should:

SORT the table

Use READ TABLE WITH KEY BINARY SEARCH for better performance.

Page: 30/63



ABAP Programming Standards

5.6.14 Sorting internal tables

When Sorting internal tables, specify the fields to Sorted.

SORT ITAB BY FLD1 FLD2.

is more efficient than

SORT ITAB.

5.6.15 Number of entries in an internal table

To find out how many entries are in an internal table use DESCRIBE.

DESCRIBE TABLE ITAB LINES CNTLNS.

is more efficient than

LOOP AT ITAB.

CNTLNS = CNTLNS + 1.

ENDLOOP.

5.6.16 Nested SELECTs versus table views/joins

Since release 4.0, OPEN SQL allows both inner and outer table joins. A nested SELECT loop may be used to

accomplish the same concept. However, the performance of nested SELECT loops is very poor in comparison   
to a join. Hence, to improve performance by a factor of 25x and reduce network load, you should either create   
a view in the data dictionary then use this view to select data, or code the select using a join.   
Example:

SELECT \* INTO GT\_WA

FROM SPFLI AS P INNER JOIN SFLIGHT AS F   
 ON P~CARRID = F~CARRID AND

P~CONNID = F~CONNID.

ENDSELECT.

is more efficient than

SELECT \* FROM SPFLI INTO GT\_SPFLI\_WA.

SELECT \* FROM SFLIGHT INTO GT\_SFLIGHT\_WA

WHERE CARRID = SPFLI\_WA-CARRID

AND CONNID = SPFLI\_WA-CONNID.

ENDSELECT.

ENDSELECT.

Due to the way that SAP has implemented JOINS it is important to follow the following general rules:

1. Do not join more than 3 tables in on a select statement

2. Ensure you join using all possible primary keys

3. Use of MANDT in Join clause: Oss note: 621640

Ensure that there is a one to one relationship between the tables being joined.

5.6.17 If nested SELECTs must be used

As mentioned previously, performance can be dramatically improved by using views instead of nested

SELECTs, however, if this is not possible, then the following example of using an internal table in a nested SELECT can also improve performance by a factor of 5x:

Use this:

FORM SELECT\_GOOD.

DATA: T\_VBAK LIKE VBAK OCCURS 0 WITH HEADER LINE.

DATA: T\_VBAP LIKE VBAP OCCURS 0 WITH HEADER LINE.

SELECT \* FROM VBAK INTO TABLE T\_VBAK UP TO 200 ROWS. SELECT \* FROM VBAP

FOR ALL ENTRIES IN T\_VBAK

WHERE VBELN = T\_VBAK-VBELN.

ENDSELECT.

ENDFORM.

Page: 31/63



ABAP Programming Standards

Instead of this:

FORM SELECT\_BAD.

SELECT \* FROM VBAK UP TO 200 ROWS.

SELECT \* FROM VBAP WHERE VBELN = VBAK-VBELN.

ENDSELECT.

ENDSELECT.   
ENDFORM.

Although using "SELECT...FOR ALL ENTRIES IN..." is generally very fast, you should be aware of the four pitfalls of   
using it:

Firstly, SAP automatically removes any duplicates from the rest of the retrieved records. Therefore, if you

wish to ensure that no qualifying records are discarded, the field list of the inner SELECT must be designed to ensure the retrieved records will contain no duplicates (normally, this would mean including in the list of retrieved fields all of those fields that comprise that table's primary key).

Secondly, if you were able to code "SELECT ... FROM <database table> FOR ALL ENTRIES IN TABLE <itab>" and the internal table <itab> is empty, then all rows from <database table> will be retrieved.

Thirdly, if the internal table supplying the selection criteria (i.e. internal table <itab> in the example

"...FOR ALL ENTRIES IN TABLE <itab>") contains a large number of entries, performance degradation may   
occur.

"SELECT...FOR ALL ENTRIES IN..." can lead to extremely long select statements, which can cause ABAP short dumps, due to limitations of the underlying database on the length of the select statement

5.6.18 ORDER BY clause

Do not use Order By clause on non-key or non-indexed fields. Bring the data into an internal table with a single call and then sort the internal table. This relieves the database of the additional sort requirement and utilizes the processing and memory of the application server.

5.6.19 HAVING clause

In a SELECT statement, the HAVING clause allows you to specify a logical condition for the groups in a GROUPBY clause. Effective use of the having clause can reduce the set of data transferred from the database to the   
application server. When the having clause is used, the aggregates and groups are constructed in the database instead of the application server, thereby reducing the resulting set.

5.6.20 SELECT \* versus SELECT individual fields

In general, use a SELECT statement specifying a list of fields instead of a SELECT \* to reduce network traffic and improve performance. For tables with only a few fields the improvements may be minor, but many SAP tables contain more than 50 fields when the program needs only a few. In the latter case, the performance gains can be substantial. For example:

Use:

SELECT VBELN AUART VBTYP FROM TABLE VBAK

INTO (VBAK-VBELN, VBAK-AUART, VBAK-VBTYP)

WHERE ...

Instead of using:

SELECT \* FROM VBAK WHERE ...

Page: 32/63



ABAP Programming Standards

5.6.21 WHERE clause

Specifying values for as many of the table’s key fields in a WHERE clause will make the SELECT statement more efficient than checking values after the select.

For example:

PARAMETERS: P\_LANGU LIKE SY-LANGU.

SELECT \*

FROM T005T

WHERE SPRAS EQ P\_LANGU.

ENDSELECT.

is more efficient than:

PARAMETERS: P\_LANGU LIKE SY-LANGU.

SELECT \*

FROM T005T.

CHECK T005T-SPRAS = P\_LANGU.

ENDSELECT.

Specifying the ‘left-most’/least specific key fields in a WHERE clause improves efficiency. For example, the key fields (in sequence) of the table KNC3 (Customer special G/L transaction)

MANDT - Client

KUNNR - Customer number

BUKRS - Company code

GJAHR - Fiscal year

SHBKZ - Special G/L indicator

When selecting data from this table, it would be more efficient to specify a value in the WHERE clause for the field KNC3-BUKRS, rather than KNC3-GJAHR. That is:

SELECT \*

FROM KNC3

WHERE KUNNR EQ „0000000001‟ AND BUKRS EQ „US01‟.

ENDSELECT.

will be more efficient than:

SELECT \*

FROM KNC3

WHERE KUNNR EQ „0000000001‟ AND GJAHR EQ „1996‟.

ENDSELECT.

You can specify as many WHERE conditions as you like in all types of database tables - i.e.

transparent tables, pool tables and cluster tables. However, you should be aware for

performance reasons that complex WHERE conditions involving pool and cluster tables usually

cannot be passed to the database system. They must be processed by the SAP database interface through post-selection.

When accessing pool and cluster tables, these should be accessed using the full primary key.   
Avoid placing a ‘SELECT’ or a ‘SELECT SINGLE’ in a loop to minimize the number of database requests.

Avoid using ‘SELECT…. Into corresponding field’ as the associated overhead with corresponding field could be significant.

Selecting via non-key fields. When selecting records from a database table when only part of a field (on   
which selection is based) is known, use the LIKE option as part of the WHERE clause.   
For example:

SELECT \*

Page: 33/63



ABAP Programming Standards

FROM T001G

WHERE BUKRS EQ „US01‟

AND TXTKO LIKE „\_\_PERS%‟.

ENDSELECT.

is more efficient than:

SELECT \*

FROM T001G

WHERE BUKRS EQ „US01‟.

CHECK T001G-TXTKO+2(4) = „PERS‟.

5.6.22 Avoid unnecessary statements

There are a few cases where one command is better than two. For example:

Use:

APPEND <TAB\_WA> TO <TAB>.

Instead of:

<TAB> = <TAB\_WA>.

APPEND <TAB> (MODIFY <TAB>).

And also, use:

IF NOT <TAB>[] IS INITIAL.

Instead of:

DESCRIBE TABLE <TAB> LINES <LINE\_COUNTER>. IF <LINE\_COUNTER> > 0.

5.6.23 Copying or appending internal tables

Use this:

<TAB2>[] = <TAB1>[]. (IF <TAB2> IS EMPTY)

Instead of this:

LOOP AT <TAB1>.

APPEND <TAB1> TO <TAB2>.

ENDLOOP.

However, if <tab2> is not empty and should not be overwritten, then use:

APPEND LINES OF <TAB1> [FROM INDEX1] [TO INDEX2] TO <TAB2>.

5.6.24 Declaring internal tables using R/3 Release 4.x syntax

See also 4.1.10 Using Internal Tables.

R/3 release 4.6 contains capabilities for hashing and sorting internal tables that improve system performance.   
The 4.6 syntax should be used whenever coding internal tables. The following examples illustrate the use of   
this syntax.

First, declare the header line or work area (wa) as a type:

TYPES: BEGIN OF <LINE\_TYPE>,

FIELD1 LIKE ...,

FIELDN LIKE ..,

END OF <LINE\_TYPE>.

For example, here is a table line type with some vendor fields

TYPES: BEGIN OF TY\_VENDOR\_DATA,

LIFNR TYPE LFA1-LIFNR,

NAME1 TYPE LFA1-NAME1, END OF TY\_VENDOR\_DATA.

or

TYPES: <LINE\_TYPE> LIKE <DD TABLE OR DDSTRUCTURE>.

This version would include ALL of the vendor (LFA1) fields

TYPES: TY\_VENDOR\_DATA TYPE LFA1.

You may then declare the internal table (see Below) or, optionally, you may first declare a table type. Declaring a table type would be preferable in those cases where your program declares several internal tables of the

Page: 34/63



ABAP Programming Standards

same type (and therefore the table type can be reused by each internal table declaration) or where the internal table is passed to subroutines or methods (and therefore the table type can be included in the form/method interface definition.

TYPES: <T\_TABLE\_TYPE> TYPE <LINE\_TYPE> OCCURS 0.

This example would define a standard (i.e. unsorted) table type

TYPES: TY\_T\_VENDOR TYPE TY\_VENDOR\_DATA OCCURS 0.

or

TYPES: <T\_TABLE\_TYPE> TYPE

[SORTED|HASHED|STANDARD|INDEX|ANY] TABLE OF <LINE\_TYPE>

WITH [UNIQUE|NON-UNIQUE] KEY <KEY DEFINITION>].

This example would define a vendor table type sorted by vendor number

TYPES: TY\_T\_VENDOR TYPE SORTED TABLE OF TY\_VENDOR\_DATA   
 WITH UNIQUE KEY LIFNR.

Finally declare the table and the work area:

DATA:<T\_TABLE> TYPE <T\_TABLE\_TYPE>,"ITAB W/ NO HEADER LINE

DATA:<WA\_TABLE>TYPE <LINE\_TYPE>. " WORK AREA FOR T\_TABLE

This example defines an itab using the table type we just defined above

Note: depending on which of the two above table types you choose this will define either a standard or sorted internal table.

DATA: T\_VENDOR TYPE TY\_T\_VENDOR, " ITAB HAS NO HEADER LINE

WA\_VENDOR TYPE TY\_VENDOR\_DATA." WORK AREA FOR T\_VENDOR

declaration of an internal table using a table line type

DATA: <T\_TABLE> TYPE

[SORTED|HASHED|STANDARD|INDEX|ANY] TABLE OF <LINE\_TYPE>

WITH [UNIQUE|NON-UNIQUE] KEY <KEY DEFINITION>]

[WITH HEADER\_LINE].

This example declares a standard, unsorted internal table

DATA: T\_VENDOR TYPE TY\_VENDOR\_DATA OCCURS 0 WITH HEADER LINE.

This example declares an internal table sorted by vendor number

DATA: T\_VENDOR TYPE SORTED TABLE OF TY\_VENDOR\_DATA   
 WITH UNIQUE KEY LIFNR.

WITH HEADER LINE.

Page: 35/63



ABAP Programming Standards

6 Naming conventions

6.1.1 Transport requests

ARCHITECTURE AND INFRASTUCTURE TEAM would publish a document specifying the details of transport request naming. Addition constraints for transport requests will be: ARCHITECTURE AND INFRASTUCTURE TEAM NAMING + Development spec ID + free text

e.g. R1GTRSYSTCI - 0012 - “Free text”

6.1.2 Package

A package per Track would be created.

6.1.3 Variable naming and declaration

ABAP/4 variable names can be up to 30 characters for DATA fields and subroutines and up to 8 characters for   
SELECT-OPTIONS and PARAMETERS, therefore, as a standard make the names descriptive. Since SAP   
segment/table-field names are hyphenated with a hyphen ('-'), use an underscore ('\_') to separate the words   
for program-specific variables. Whenever possible, the LIKE parameter should be used to define work fields.

Some variable types should be prefixed with a specified letter or letters (a distinction is made between global and local variables):

Global variables:

|  |  |
| --- | --- |
| **Global variables**: |  |
| **Variable Type** | **Prefix** |
| Selection screen parameter | p\_ |
| Select-options | s\_ |
| Internal tables | gt\_ |
| Constants | gc\_ |
| Work field | gw\_ |
| Field symbols | gfs\_ |
| Types | gty\_ |

Variable Type Prefix

Selection screen parameter p\_

Select-options s\_

Internal tables gt\_

Constants gc\_

Work field gw\_

Field symbols gfs\_

Types gty\_

Local variables:

|  |  |
| --- | --- |
| **Local variables**: |  |
| **Variable Type** | **Prefix** |
| Form routine parameter | lp\_ |
| Internal tables | lt\_ |
| Constants | lc\_ |
| Work field | lw\_ |
| Field symbols | lfs\_ |
| Types | lty\_ |

Variable Type Prefix

Form routine parameter lp\_

Internal tables lt\_

Constants lc\_

Work field lw\_

Field symbols lfs\_

Types lty\_

ABAP Objects

|  |  |
| --- | --- |
| **ABAP Objects**: |  |
| **Variable Type** | **Prefix** |
| ABAP Object Class | lcl\_ |
| Global ABAP Object | go\_ |
| Local ABAP Object | lo\_ |

Variable Type Prefix

ABAP object class lcl\_

Global ABAP object go\_

Local ABAP object lo\_

Page: 36/63



ABAP Programming Standards

When declaring internal variables, use the LIKE option only for program internal objects. Refer to ABAP

Dictionary Objects using TYPE since these are data types. Use the reference objects in the ABAP Dictionary whenever possible to ensure that the variable has the same field attributes as the existing object, which has already been declared. If the type of field, to which you are referring, changes the ABAP runtime system   
updates all references automatically. It also stops the system from carrying out unnecessary (and maybe   
undesirable) type conversions.

Example: DATA: LIFNR TYPE LFA1-LIFNR.

**Ranges:**

Ranges are obsolete and hence should not be used.

Following syntax may be used which will declare an internal table with similar structure of ranges.

DATA: LT\_MARA TYPE RANGE OF MATNR.

6.2 ABAP development

6.2.1 ABAP Programs

The maximum length of an ABAP program name (SE38) is 30 characters.

Format: Z XXXXXXXXXXXXXXXXXXXXXXXXX

with 2 26 X Meaningful description

6.2.2 Module Pools & Includes

Module pools should be named as follows:

Format: SAPM Z XXXXXXXXXXXXXXXXXXXXXX

with 1 4 SAPM required by SAP to denote online modules

5 Z Z to denote customer named object

6 27 X Meaningful description

Page: 37/63



ABAP Programming Standards

Module pool includes should be named as follows:

Format: M Z XXXXXXXXXXXXXXXXXXXXXX T NN

With 1 M required by SAP to denote online modules

2 Z Z to denote customer named object

3 24 X Meaningful description corresponding to main program

25 T I PAI modules

O PBO modules

F subroutines

26 27 N Sequential number (00 to 99)

For the global data include, T and N are replaced by ‘TOP’.

6.2.3 Transaction Codes

All SAP transactions have unique transaction codes, which are listed in tables TSTC and TSTCP. Transactions are maintained via transaction SE93. Transaction codes consist of up to twenty characters.

Format: Z XXXXXXXXXXXXXXXX

with: 2 17 X Meaningful description

6.2.4 Dynpros (Screens)

SAP screens are referred to as "Dynpros". Standard SAP components, such as transactions, menus and tables, contain Dynpros and the associated processing logic. The user can also generate customised Dynpros. (SE51 -  
Screen painter)

The identification of a Screen Painter Dynpro consists of an ABAP program name (30 characters) and a fourdigit screen number.

Format: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA NNNN

with: 1 30 A Associated ABAP program name (See naming convention for ABAP

programs above)

31 34 N Screen number (number in intervals of ten, starting at 9000)

Page: 38/63



ABAP Programming Standards

6.2.5 Dialog Modules

The maximum length of a dialog module name (SE35) is 30 characters.

Format Z XXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 27 X Meaningful description

6.2.6 Messages

Messages (SE91) are referenced in ABAP programs by a twenty-character message class and a 3-character

message ID (preceded by the message type identifier).

Format: Z XXXXXXXXXXXXXXXX NNN

with: 2 17 X Meaningful description

18 20 N Message number (sequentially assigned number)

NB: The message number ‘000’ should not be used.

6.2.7 Function Groups

Function group names are twenty-six characters in length. One function group should be created per table

view maintenance screen, to ease potential transport problems.

Format: Z XXXXXXXXXXXXXXXXXXXXX

with: 2 22 X Meaningful description

6.2.8 Function Modules

The maximum length of a function module name (SE37) is 30 characters.

Format Z\_ XXXXXXXXXXXXXXXXXXXXXXXXX

with: 3 27 X Meaningful description

Page: 39/63



ABAP Programming Standards

6.2.9 Logical Databases

A logical database creates a logical hierarchical view of several physical relational tables, giving the ability to link   
database tables, thus simplifying report programming through standardised read access, authorization checks   
and selections.

Logical database names consist of twenty characters.

Format: Z XXXXXXXXXXXXXXXXXX

with: 2 19 XX Meaningful description

6.2.10 Business Add-Ins

Business Add-Ins are a new SAP enhancement technique based on ABAP Objects. They can be inserted into the SAP System to accommodate user requirements too specific to be included in the standard delivery

BADI implementation names consist of twenty characters.

Format: Z XXXXXXXXXXXXXXXXXX

with: 2 19 XX Meaningful description (ref. the BADI definition if possible)

Page: 40/63



ABAP Programming Standards

6.3 ABAP Objects Programming

Note that we don’t usually include a scope element in the name of the objects described in this section. This is for a number of reasons:

The scope of a class component is explicitly defined as part of the declaration of the method -  
there is no need to specify it in the name,

The traditional scope ‘buckets’ (global, local, etc.) don’t map well to the OO paradigm,

OO object and component names are supposed to be fixed and relevant to their responsibility - it would be absurd to have to rename a method because its scope changed if its functionality   
remained the same.

6.3.1 Classes (defined in DDIC or programmatically)

A class is a template for an encapsulated piece of data and functionality that represents a physical or logical object. It could be described as an object’s type.

The maximum length of a class name is 30 characters. The name may consist of alphanumeric characters plus the special characters underscore (\_) and forward slash (/). The forward slash (/) is used as a delimiter for the namespace prefix. The name may not begin with a digit.

Format: ZCL\_XXXXXXXXXXXXXXXXXXXXXX

with: 5 26 X Meaningful description.

NB: Strictly, the descriptive part of a class name should be a noun. Note that SAP rarely follows this OO naming convention, since they have restricted the length of the class name in the DDIC.

6.3.2 Interfaces (defined in DDIC or programmatically)

The components of a class are divided into visibility sections, and this forms the external point of contact

between the class and its users. For example, the components of the public section form the public interface   
of the class, since any user can access all attributes and method parameters. The protected components form   
an interface between the class and those classes that inherit from it (subclasses). Interfaces are independent   
structures that allow you to enhance the class-specific public points of contact by implementing them in   
classes. Different classes that implement the same interface can all be addressed in the same way. Interfaces   
are the basis for polymorphism in classes, because they allow a single interface method to behave differently   
in different classes. Interface references allow users to address different classes in the same manner.   
Interfaces can also be nested.

The maximum length of an interface name is 30 characters. The name may consist of alphanumeric characters plus the special characters underscore (\_) and forward slash (/). The forward slash (/) is used as a delimiter for the namespace prefix. The name may not begin with a digit.

Format: ZIF\_XXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 30 X Meaningful description.

Page: 41/63



ABAP Programming Standards

NB: Strictly, the descriptive part of an interface name should be an adjective that describes the ability that a class will gain if it properly implements the interface. E.g. handleable, deletable. Note that SAP rarely follows this OO naming convention, since they have restricted the length of the interface name in the DDIC.

6.3.3 Methods (defined in DDIC or programmatically)

Methods are the internal procedures of a class that determine the behaviour of an object. They can access all the attributes of their class and can thus change the object status. Methods have a parameter interface,   
through which the system passes values to them when they are called, and through which they can return values to the caller. The private attributes of a class can only be changed using methods. In terms of definition and   
passing parameters, methods are similar to the old function modules.

Format: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 1 30 X Meaningful name.

NB: Method names should principally be verbs describing what will happen when the method is invoked. E.g. handle\_event, destroy, get\_page\_url. Separate words with underscores, since SAP is not case sensitive here.

6.3.4 Attributes (defined in DDIC or programmatically)

Attributes are internal data fields of any ABAP data type within a class. The content of the attributes specifies the status of the object. You can also define reference variables, which you can then use to create and address objects. This allows objects to be accessed within classes.

Format: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 1 30 X Meaningful name.

6.3.5 Events (defined in DDIC or programmatically)

Events allow an object or class to trigger event handler methods in another class or object. Just as, in a normal   
method call, a method can be called by any number of users, any number of event handler methods can be   
called when an event is triggered. The trigger and handler are coupled at runtime. In a normal method call, the   
caller specifies which method it wants to call; the relevant method must be available. With events, the handler   
specifies the events to which it wants to react; a handler method must be registered for each event.

Format: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 1 30 X Meaningful name.

NB: Event names should principally be past tense verbs describing what happened to cause the event to be   
raised. E.g. button\_clicked, window\_closed. Separate words with underscores, since SAP is not case sensitive   
here.

6.3.6 Internal Types (defined in DDIC or programmatically)

Types define your own ABAP data types within a class. Use the TYPES statement to declare constants Types are not instance- specific - they are available once and once only for all the objects in the class.

Format: TY\_XXXXXXXXXXXXXXXXXXXXXXXXX

with: 4 30 X Meaningful name.

6.3.7 Constants (defined in DDIC or programmatically)

Constants are special static attributes, whose values are specified when they are declared and which cannot be changed later. Use the CONSTANTS statement to declare constants. Constants are not instance-specific - they are available once and once only for all the objects in the class.

Format: C\_XXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 3 30 X Meaningful name.

Page: 42/63



ABAP Programming Standards

6.4 Data Dictionary

6.4.1 Domains

A domain is a central object for describing the attributes of a business object. A domain describes the physical value set for a field. This set of values is defined by specifying formal attributes, such as external format,   
length, etc., which are entered as individual values or in the form of a table.

The maximum length of a domain name is 30 characters.

Format: Z XXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 30 X Meaningful description

6.4.2 Data Elements

A data element is a semantic domain. It gives a precise description of the function of a domain in a specific business context for the benefit of the fields dependent on it.

The maximum length of a data element name is 30 characters.

Format: Z XXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 30 X Meaningful description

6.4.3 Tables

The following section describes the requirements for the different types of tables and fields within them. Data (transparent) Tables:

The maximum length of a standard SAP table name is 16 characters.

Format: Z XXXXXXXXXXXXXXX

with: 2 16 X Meaningful description

Table Fields:

The maximum length of a table field is 16 characters.

Format: XXXXXXXXXXXXXXXX

with: 1 16 X Meaningful description

This applies to all table fields in custom/bespoke tables.

6.4.4 Append Structures

Append structures are used for adding customer fields to the standard SAP tables, and protect these additional fields from being overwritten by SAP upgrades. Whenever possible, an existing SAP append structure should be extended. If the table has no appends, the system proposes a standard name. This, too, should be used whenever possible. However, if a custom name is to be created, the following conventions apply.   
The maximum length of the name of an append structure is thirty characters. It has to be created in the   
customer name range, even though it is an extension to an original SAP table.

Format: Z XXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 30 X Name of the table to which the structure is appended

plus a numeric identifier

Fields in an Append Structure:

The maximum length of an append structure field is thirty characters.

Format: Z XXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 30 X Meaningful description

6.4.5 Views

A view is a virtual table (not containing any data). It presents data from one or more related tables in a tabular form appropriate for application processing.

The maximum length of a view name is 16 characters.

Format: Z XXXXXXXXXXXXX

with: 2 14 X Meaningful alphanumeric identifier

View Fields:

Page: 43/63



ABAP Programming Standards

The maximum length of a view field is 10 characters.

Format: XXXXXXXXXX

with: 1 10 X Meaningful description

6.4.6 Structures

Structures are global definitions of the structure of data used during calculation operations within programs, or when passing data between programs.

The maximum length of a structure name is 30 characters.

Format: Z XXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 28 X Meaningful description

Structure Fields:

The maximum length of a structure field is 30 characters.

Format: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 1 30 X Meaningful description

6.4.7 Search Helps (Matchcodes)

A search help (matchcode before Release 4.0) is a search criterion enabling efficient access to records without requiring knowledge of the key term.

Collective Search Help

Collective search helps (equivalent to release 3.x matchcode objects) have a maximum of thirty characters, with the first two characters being the Entity Code, and the remaining characters meaningfully assigned.

Format: Z XXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 28 X Meaningful description

Elementary Search Help

All data access is done via an elementary search help (equivalent to a release 3.x matchcode ID). A number of elementary search helps can be assigned to one collective search help.

The name of an elementary search help can be up to thirty characters in length.

Format: Z XXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 28 X Meaningful description

6.4.8 Lock Objects

Lock objects are used to lock dictionary objects during a logical unit of work, e.g. until the next COMMIT is

reached. The maximum length of a lock object name is 16 characters.

Format: E Z XXXXXXXXXXXXX

with: 1 E Constant 'E'

3 15 X Meaningful description

6.5 ALE/IDOC Developments

6.5.1 IDOC Segment Types (version-independent)

Format: Z XXXXXXXXXXXXXXXXXXXXXXX I

with: 2 24 X Meaningful description

25 I 0-9, sequential number to avoid conflict

6.5.2 Basic IDOC Types

Format: Z XXXXXXXXXXXXXXXXXXXXXXXX II

with: 2 25 X Meaningful description

26 27 I 01-99, sequential number to avoid conflict

Page: 44/63



ABAP Programming Standards

6.5.3 Extension Types

Format: Z XXXXXXXXXXXXXXXXXXXXXXXX II

with: 2 25 X descriptions of the customized IDOC being extended

26 27 I 01-99, sequential number to avoid conflict

6.5.4 Message Types

Format: Z XXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 27 X meaningful description, relating to the IDOC this message type is for

6.5.5 Process Codes

Format: Z XXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 27 X meaningful description, relating to the IDOC this process code is for

6.5.6 Inbound ALE Function Modules

Format: ZIDOC\_INPUT\_XXXXXXXXXXXXXXXXXX

with: 13 30 X message type description

6.5.7 Outbound ALE Function Modules

Format: ZIDOC\_OUTPUT\_XXXXXXXXXXXXXXXXX

with: 14 30 X message type description

6.5.8 Business Object and BAPI developments

6.5.9 Business Object Type (within the BOR)

Format: Z XXXXXX I

with: 2 7 X meaningful description

8 I 0-9, sequential number to avoid conflict

6.5.10 Business Object Method

Format: xxxxxxxxxxxxxxxxxxxx

with: 1 20 x short description, case sensitive (e.g. Change Password)

6.5.11 BAPI based function module

Format: ZBAPI\_XXXXXXXXX\_YYYYYYYYYYYYY

with: 7 16 X Business Object Name

17 30 Y Method Name

6.6 Workflow developments

The following is a list of general rules for Workflow object component names:

Short Description should use uppercase letters on the first letter of each abbreviated word and lower case   
letters for the remainder of each abbreviated word. E.g. ChgMstr (with no spaces between abbreviated   
words)

Page: 45/63



ABAP Programming Standards

Underscores should be avoided if possible, and uppercase/lowercase used instead (within a work as well) to improve readability.

Use of upper case/lower case does not affect identification; in other words, object component names are not case sensitive.

Names should be nouns and begin with an uppercase letter.

SAP Object Len Naming Conventions

Object Type 10 ZWO + ShortDesc

O Object

ShortDesc Short description of object name (7

chars)

If an object type is defined as a business object

(BUSnnnn), the same name as that of the assigned data model should be chosen

If not a business object, the 7 characters should use the following format:

e.g. ZWOAAAnnnnn

AAA Application area

NNNNNN Sequential number

e.g. Purchase Requisition - ZWOBUS2009

Object Type - Interfaces 10 ZIF + ShortDesc

IF Interface

ShortDesc Short description (6 chars)

e.g. ZIFSAP

Object name 32 Z + ShortDesc + I

ShortDesc Short description (up to 30 chars) I

0 - 9 sequential number

Object - Key Fields 32 K + ShortDesc

K Key Fields

ShortDesc Short description (31 chars)

e.g. KPurchReqNo

Object - Attributes 32 AAAb + ShortDesc

A Attribute

a D  Database field

V  Virtual field

b S  Single line

M  Multi-line

I  Instance independent O  Only modelled

ShortDesc Short description (29 chars) e.g. ADSCostCentre

Object - Methods 32 Mabc + ShortDesc

M Method

Page: 46/63



ABAP Programming Standards

SAP Object Len

Object - Method - Function 30   
Modules

Object - Method - BAPI 30   
Function

Object - Method - ABAP/4 30

Reports

Object - Method - Dialog 30 Modules - ABAP/4

Object - Events 32

Object - Implementation 30   
Program

Standard Tasks 12

Workflow Templates 12

Naming Conventions

a S  Dialog synchronous

A  Dialog asynchronous B  Background

b F Function module

R  ABAP report   
T  SAP transaction   
D  Dialog method   
B  API method

O  Other

c P  Parameter(s)

N  No parameter(s)

ShortDesc Short description (28 chars) e.g. MSTNCreatePurchReq

ZWF\_ + ShortDesc

ShortDesc Short description of method (19 chars) ZBAPI\_<object type>\_<method>

<object type> Object type of this BAPI in BOR   
<method> Method name of this BAPI in BOR ZWR + ShortDesc

W Workflow   
R ABAP Report

ShortDesc Short description (20 chars) ZWD + ShortDesc

W Workflow

D Dialog Module - ABAP   
ShortDesc Short description (20 chars) WE + ShortDesc

W Workflow   
E Event

ShortDesc Short description of event (30 chars) e.g. WEPurchReqApproved

ZW + <object type>

ZWTSa+ ShortDesc

TS Standard task (single step)

a D  Dialog

B  Background

ShortDesc Short description (7 chars) e.g. ZWTSDApprReq

ZWWSa + ShortDesc

Page: 47/63



ABAP Programming Standards

SAP Object Len

Container Elements (field 32   
names)

Standard Roles 12

Standard Roles: Function 30   
module

Organisation Unit 12

Org Units - Jobs 12

Org Units - Positions 12

Naming Conventions

WS Workflow template   
 (multi-step)

a SAP Module

ShortDesc Short description (7 chars) e.g. ZWWSMReqRel

Cab + ShortDesc

C Container Element

a W  Workflow tasks & templates

T  Standard/Customer task R  Role resolution

b O  Object type

D  DDIC field

ShortDesc Short description (29 chars) e.g. CTDPurchReqNo

ZWRa + ShortDesc   
R Standard Role

a F Function module

O Organizational object R Responsibilities

ShortDesc Short description (8 chars) e.g.. ZWRF\_REQUEST

ZWF\_ROLE + ShortDesc

ShortDesc Short description (15 chars) e.g. ZWF\_ROLEGetRequisition

ZWUO + ShortDesc

U Organisational Structure

O Organization Unit

ShortDesc Short description (8 chars)

e.g. ZWUOPurch

Org. unit is needed if not using HR. If HR is being implemented, all PD org. objects and their names should be coordinated with the HR team

ZWUP + ShortDesc

U Organizational Structure

P Job

ShortDesc Short description (8 chars) e.g. ZWUPBuyers

ZWUS + ShortDesc

U Organizational Structure

S Position   
ShortDesc Short description (8 chars)

Page: 48/63



ABAP Programming Standards

SAP Object Len Naming Conventions

e.g. ZWUSRawMatrl

6.7 Business Server Page (BSP) Applications

6.7.1 BSP Application Object

Format: Z\_AAA\_XXXXXXXXXXXXXXXXXXXXXXXXX

with: 3 4 5 A Application Area

6 30 X Meaningful description

6.7.2 Controller

Format: Xxxxxxxxxxxxxxxxxx.do

with: 1 67 Xx Meaningful description (Case sensitive, so use keyword

capitalisation - no underscores)

6.7.3 BSP page (View, Page with Flow Logic, or Page Fragment)

Format: Xxxxxxxxxx.htm or Xxxxxxxxxx.bsp

with: 1 67 Xx Meaningful description (Case sensitive, so use keyword

capitalisation - no underscores)

6.7.4 Page attribute

Format: Xxxxxxxxxx.htm or Xxxxxxxxxx.bsp

with: 1 67 Xx Meaningful description (Case sensitive, so use keyword

capitalisation - no underscores)

6.7.5 MIME object

MIME objects are usually imported, so the suffix will follow the convention of the imported object.

6.8 SAPSCRIPT Development

6.8.1 Layout Sets

Layout sets are SAP output forms that are defined using SAPSRIPT. (SE71)

The layout set name can be a maximum of sixteen characters.

Format: Z AAA XXXXXXXXXXXX

with: 2 3 4 A Application Area

5 16 X Meaningful description

6.8.2 Styles

Styles are character and paragraph definitions to be used in layout sets. (SE72)

Format: Z XXXXXXX

with: 2 8 X Meaningful description

Page: 49/63



ABAP Programming Standards

6.8.3 Standard Texts

Standard texts are SAP text modules to be used in various documents. (SO10)

They consist of a text name (up to thirty-two characters long) assigned to a text ID. Format (Text ID): Z XXX

with: 2 4 X Meaningful alphanumeric identifier

Format (Text name): Z\_ AAA XXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 3 4 A Application Area

4 32 X Meaningful description

6.9 Authorizations

6.9.1 Authorization Objects

Authorization objects may be created to apply additional security. An object has up to ten characters. The

first should be a ‘Z’, the second an underscore and the remaining eight forms a meaningful description.

Format: Z\_ XXXXXXXX

with: 1 2 C Constant 'Z\_'

3 10 X Meaningful description

6.9.2 Authorization Groups

Authorization groups may be created to group together programs for authorization checking. The

authorization group should be specified in the program attributes.

Format: Z AAA XXXXX

with: 2 3 4 AAA Application Area

5 8 X Meaningful description

6.9.3 Authorization Object Classes

Authorization object classes group authorization objects with common character. In standard SAP a functional module does this. A user-defined authorization object class should be an extension of a delivered one - e.g. any additional FI authorization objects would be placed in an additional FI authorization object class. Therefore the ‘extension’ classes should be named in a similar way to the standard delivered classes.

Format: Z XXX

with: 2 4 X three letters of original SAP class (e.g. 'MMB')

Remarks: SAP Security objects are hard-coded into the standard transactions. It is therefore very

difficult to add security effectively. Careful analysis should be made before creating any new authorization objects or classes.

6.10 Legacy System Migration Workbench (LSMW)

6.10.1 Projects

It is envisaged that one project will be created for each Track.

Format: Z AAA XXXXXXXXXXXX

with: 2 3 4 A Application Area

5 15 X Meaningful description (e.g. MIGRATION)

6.10.2 Sub-projects

One sub-project will be created for each phase.

Format: Z AAA PHASE X

with: 2 3 4 A Application Area

9 X Phase Number (i.e. 1, 2, 3 etc)

6.10.3 Object

The name of the data entity.

Format: Z XXXXXXXXXXXXXX

Page: 50/63



ABAP Programming Standards

with: 2 15 X Entity Name, followed by a sequence number (01, 02, etc)

The sequence number is there in the case where there are migration variations within one data entity.

6.11 ABAP Query

6.11.1 User Groups

The name of the user group.

Format: Z AAA XXXXXXXXX

with: 2 3 4 A Application Area

5 12 X Meaningful description

6.11.2 InfoSets

The name of the InfoSet

Format: Z AAA XXXXXXXXXXXXXXXXXXXX

with: 2 3 4 A Application Area

5 24 X Meaningful description

6.11.3 Queries

The name of the Query

Format: Z AAA XXXXXXXXXXX

with: 2 3 4 A Application Area

5 14 X Meaningful description

Page: 51/63



ABAP Programming Standards

7 Documentation standards

ABAP/4 code is fairly self-documenting. However, it is wise to provide future programmers with

documentation. Explain the purpose at the top of the program. Maintain a history of modification notes,

dated, with the most recent change first. Comment work fields, and fields in work records especially those used for interfacing. Comment all subroutines with their purpose. Comments should explain what the code is doing, not how the code is doing it.

7.1 Header Documentation Box

The Header comment box similar to the one below should be coded after the REPORT, or after FUNCTION

MODULE statement of customised ABAP/4 programs both for new and cloned programs. The documentation

box must be used to list detailed information from functional and technical specifications as well as

documenting modifications to the program.

\*-----------------------------------------------------------------------\*

\* Program Name : ZMUKLOAD

\* Title : Upload Program for Material Data

\* Create Date : 28-Feb-2002

\* Release : 6.0

\* Author : LROGERS

\*-----------------------------------------------------------------------\*

\* Description : This program is used to upload the Material data from

\* the UNIX file.

\*-----------------------------------------------------------------------\*

\* CHANGE HISTORY

\*-----------------------------------------------------------------------\*

\*Date | User ID |Description |Change Label \*

\*-----------------------------------------------------------------------\*

\* 8-Mar-2002| PJOHNS | Added a new fields to the | PJ01 \*

\* | | report output. | \*

\* | | | \*

\* | | | \*

\*-----------------------------------------------------------------------\*

A standard comment box can be created and maintained as a pattern via “UtilitiesMore

UtilitiesModeCreateChange pattern” in the ABAP/4 editor. Once set-up the comment box can be   
added to the code by entering `IC xxxx` in the command field of the ABAP/4 editor where xxxx is the comment   
box identifier.

After the program has been transported to production further changes to the code should be marked with a   
change-label. This can be of the form “XXXNN” where XXX are the initials and NN is a number. Coding lines to be   
deleted should be commented out and marked with the change-label and change label should be

mentioned as reference in the change history in the header.

EXAMPLE.

WRITE: VBAK-VBELN, “PJ01

VBAK-VBELP. “PJ01

Page: 52/63



ABAP Programming Standards

After a period of time the commented out (deleted) lines should be removed from the coding to avoid the

program having too many unnecessary lines and to avoid confusion.

7.2 Comment Box

A comment box is used to emphasise a statement or section in a program.

Example:

\*--------------------------------------\*

\* LOGIC SECTION \*

\*--------------------------------------\*

A comment box should be inserted before all: declarative elements

Events.

Example:

\*--------------------------------------\*

\* GLOBAL DATA \*

\*--------------------------------------\*

DATA: W\_KUNNR, "Customer nr

W\_MATNR. "Material nr

\*--------------------------------------\*

\* TOP-OF-PAGE \*

\*--------------------------------------\* TOP-OF-PAGE.

PERFORM WRITE\_HEADER.

Other comment boxed which must be used are:

\*--------------------------------------\*

\* SELECT-OPTIONS / PARAMETERS \*

\*--------------------------------------\*

Before the declaration of parameters and select-options.

\*--------------------------------------\*

\* DATA DECLARATION \*

\*--------------------------------------\*

Before the data declaration part.

\*--------------------------------------\*

\* LOGIC SECTION \*

\*--------------------------------------\*

Before the start of the main program code. Mostly before START-OF-SELECTION.

7.3 Comment box before FORM/METHOD declaration

A declaration box before a FORM should be used. Comments and parameter explanation can be added here. Use the standard box as produced by SAP when double clicking on the PERFORM line to create the FORM.

Example:

\*&---------------------------------------------\*

\*& Form WRITE\_HEADER \*

\*&---------------------------------------------\*

\* Write list header \*

\*----------------------------------------------\*

\* --> p\_kunnr Customer number \*

\* <-- rcode return code \*

\*----------------------------------------------\* FORM WRITE\_HEADER USING KUNNR

CHANGING RC.

Page: 53/63



ABAP Programming Standards

ENDFORM. " WRITE\_HEADER

7.4 Comment lines

There are two ways of indicating comments in a report:

1. If the whole line is supposed to be a comment, enter an asterisk (\*) in the first   
column.

Example:

\* Store Date in YYYYMMDD format in LONG DATE MOVE: SY-DATUM+0(4) TO LONG\_DATE-YEAR,

SY-DATUM+4(2) TO LONG\_DATE-MONTH,   
SY-DATUM+6(2) TO LONG\_DATE-DAY.

Comment lines can be inserted anywhere in a report and so this should be used to describe the function of certain parts of the code.

2. If only the latter part of a line is supposed to be a comment, enter “ before the comment.   
Example:

TABLES: TOO1, "SAP Company Code Tables

TOO1C, "Cost Centre Accounting Control Table CSKS. "Cost Centre Master

The system interprets comments indicated by “ as blank characters. This option should be used to describe certain lines of code.

Page: 54/63



ABAP Programming Standards

8 Special consideration

8.1 Internationalisation and Localisation Aspects

8.1.1 General Remarks and Concepts in ABAP

Different countries use different regional settings e.g. for date, decimal point and paper size. However, the   
different regional settings in different countries still apply. For example, the decimal separator in Germany,   
France, and many Spanish speaking countries (including Chile) is a comma (,) and the thousand separator is a

(.). So 1,000 would be interpreted in these countries as one rather than one thousand (represented as

1.000,00). If a SAP system started producing reports or screen displays of numbers in a single format

worldwide, this could lead to major confusion to users and others (e.g. customers) reading printed reports or   
screen displays in many countries. Another obvious example is dates. In the US, the date format is   
MM/DD/YYYY, while the rest of the world is generally on DD/MM/YYYY. 01/03/2001 therefore needs to be   
interpreted as 1 March 2001 in most of the world, but as 3 January 2001 in the United States.   
Regional settings may be different for different countries speaking the same language, so is not necessarily   
unique to a language. For example, Switzerland speaks French, German and Italian and has a dot (.) as the   
decimal separator. But in France (also speaking French) or Germany (also speaking German) the decimal   
separator is a comma (,).

A SAP system has several features for internationalization/localization purposes. Not all of them are explicitly listed in this document. Some examples related to programming are listed below.

Our approach to realize country specific standards is rather pragmatic to minimize programming efforts and realize a solution feasible with the standard system. For decimal notation and date format our   
recommendation is to make it mandatory for all users to adjust the user master record using the SAP menu 'System -> User profile -> Own data' in accordance with local custom.

Not every country specific format can be found here. . E.g. 1'234'567.89, using an apostrophe (single quote) as   
separator like it is common in Switzerland is not possible. If this is really needed, it is possible in principle to   
read the front-end settings from the Registry using function module 'GUI\_GET\_REGVALUE'.   
However for reports running in background there is no front-end connection. Transferring front-end settings   
from the user scheduling the job to a standard batch user means a lot of programming effort. The pragmatic   
approach is to have a batch-user for every country specific setting in accordance with local custom.

8.1.2 ABAP Dictionary

Database tables should not have free-text-fields with language specific text, but should have a language key field and a language table where these texts can be translated.

Database tables containing amounts of money, measures of length, areas, weight, etc should have fields of   
type CURR (currency fields) and QUAN (quantity fields). For fields of type CURR a field of type CUKY (currency   
key, referenced by CURR fields) is mandatory to store the corresponding currency e.g. GBP for the amount of   
money. Similar to this there is a type UNIT (unit key) for QUAN fields specifying the unit for a quantity.

8.1.3 ABAP/4 Programming

For list creation in ABAP/4 there are similar features.

Text should never be hard-coded in the program source code but referred to as text-symbols, selection texts, header lines, messages, etc all of them can be translated to any of the imported languages. There is an   
extended syntax check on that.

The formatting options for the WRITE statement allow specifying the currency using the CURRENCY   
addition. The number of decimal places for a currency key is stored in table TCURX. If there is no   
value for the specified key, the system assumes that the currency amount has two decimal paces.

Page: 55/63



ABAP Programming Standards

Units can be set according to entries in table T006 using the UNITS statement for fields of type P. If this table does not contain the specified unit key, this option does not have an effect.

Setting date format and decimal character

Every user can specify in his/her master record (User defaults) how the date is to be formatted on output and whether the decimal character should be a comma or a period.

You can also set these output parameters for a particular program by using the statement SET COUNTRY f. This displays the decimal point and date in all subsequent output (WRITE) according to the settings specified in the table T005X for the country ID f. If he country does not exist in table T005X, the formats used are "." for the decimal point and "MM/DD/YYYY" for the date.

The special form SET COUNTRY SPACE (or f contains SPACE) resets the decimal point and date display formats to the setting contained in the current user's master record. In this case, table T005X is not read and the return code value is always 0.

The effect of SET COUNTRY is not restricted to the current program, but applies at once to all programs in the current roll area.

When downloading lists to Excel the settings mentioned above will be effective. Depending on your Excel country version and the existing number, date and time format codes the download will only interpreted as a date, time or number figure if this format exists in your Excel version. If the format code does not exist the values will be treated as characters and cells will be general format cells with no specific number format.

Paper size: please make sure that your list will fit the main paper sizes A4 (21 cm x 29.7 cm) and Letter (21.59 cm x 27.94 cm), i.e. does not exceed 21 cm x 27.94 cm.

8.1.4 SET LOCAL LANGUAGE statement

The ABAP statement SET LOCAL LANGUAGE lg allows writing programs, which work on data of different languages in a controlled manner.

Note: The effect of SET LOCALE is not restricted to the current program, but affects all programs in the current   
roll area.

The TRANSLATE statement is dependent on the current language environment. The data to be processed may   
have been entered in another language environment. E.g. when German data is processed in a Russian   
environment

DATA v\_letter(1) TYPE C value 'ö'.

TRANSLATE v\_letter TO UPPER CASE.

will assign '¦' to v\_letter. In a German environment one would get 'Ö'.

8.1.4.1 Solution 1

To avoid such an error the statement SET LOCAL LANGUAGE lg has to be used. Example: An internal table with a language key has to be processed:

DATA: BEGIN OF rec\_data OCCURS 0,

langu LIKE sy-langu,

txt(20) TYPE CHAR, END OF rec\_data.

SET LOCALE LANGUAGE rec\_data-langu. TRANSLATE rec\_data-txt TO UPPERCASE. SET LOCALE LANGUAGE SPACE.

This is only a simple solution, since it does not require you to make extensive program changes. You call each TRANSLATE TO UPPERCASE in the appropriate environment. The command SET LOCALE LANGUAGE SPACE always reverts to the logon language

Page: 56/63



ABAP Programming Standards

Unfortunately it has certain disadvantages:

The SET LOCALE command is time-consuming.

It is possible that the language contained in data-txt is not allowed. This would trigger a RABAX (exceptions can be tackled using the CATCH statement).

The program might not run in the logon language.

SET LOCALE LANGUAGE lg eventually calls the C-function setlocale, which is expensive. The price differs with language and with platform but requires several microseconds. To process a huge amount of data the number of executions should be limited by rearranging the data.

8.1.4.2 Solution 2

If sorting is no problem, use SORT to keep the number of switches to a minimum. SORT I\_DATA BY LANGU.

LOOP AT I\_DATA INTO REC\_DATA.

IF REC\_DATA-LANGU NE SY-LANGU.

SET LOCALE LANGUAGE REC\_DATA-LANGU.   
ENDIF.

TRANSLATE REC\_DATA-TXT TO UPPER CASE. ENDLOOP.

SET LOCALE LANGUAGE SPACE.

SORT DATA BY KEYFIELD1 KEYFIELD2.

8.1.4.3 Solution 3

If the data is not sorted and sorting is a problem, but the data can be quickly processed one could do several   
runs.

Create a small table with a language field. Do a first run in the logon language processing only the

corresponding data and use COLLECT to store all languages in the language table. Looping over this internal table you switch to the next language and process the corresponding datasets by running again throw the whole data, etc.

8.1.4.4 Solution 4

Process only the data corresponding to your logon language, if this does make any sense.

8.1.5 Unicode

In a Unicode system the character data is encoded in only one codepage. There is only one view and misinterpretation is not possible

8.1.5.1 Representation of Unicode Characters

The database may contain the same byte representation or any 1:1 transformation into any other Unicode representation (UTF-8, UTF-16 Big Endian, UTF-16Little Endian).

UTF-16 - Universal Transformation Format, 16 bit encoding Fixed length, 1 character = 2 bytes

Platform dependent byte order

2 byte alignment restriction

UTF-8 - Unicode Transformation Format, 8 bit encoding Variable length, 1 character = 1 - 3 bytes

Platform independent

No alignment restrictions

Page: 57/63

 ABAP Programming Standards

8.1.5.2

Unicode Restrictions

With release 6.10/6.20 SAP introduces new additions to existing key words to improve security, maintainability

and readability of ABAP programs. To minimize the efforts and costs necessary to migrate from Non-Unicode

to Unicode, programs should already now be designed in a way that takes these changes into account.

8.1.5.2.1

String Processing

With Unicode you will have to distinguish between character processing and byte processing.

Character processing: string operations are only allowed for character-like operands

ABAP types C, N, D, T and STRING

Structures consisting only of characters (C, N, D, T)

X and XSTRING are no longer considered character-like types

Byte processing: byte operations are only allowed for operands of type X or XSTRING

Addition 'IN BYTE MODE' for statements

Prefix 'BYTE-' for comparison operations

8.1.5.2.2

Length and Distance

The above-mentioned must be taken into account when using the statement DESCRIBE. With Unicode you

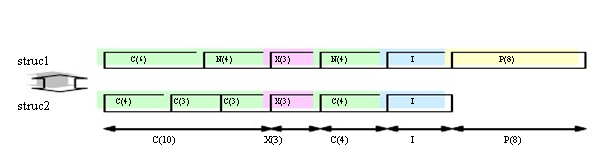
have to distinguish if you want to count the length and distance in bytes or in characters.

8.1.5.2.3

MOVE –Statement

1. MOVE between incompatible Structures

To move between structures a matching data layout is required.



Example:

DATA: BEGIN OF CSTRUC,

FIRST(10)TYPE C,

TAB(1) TYPE C,

LAST(10)TYPE C,

END OF CSTRUC.

DATA: BEGIN OF XSTRUC,

FIRST(10)TYPE C,

TAB(1) TYPE X VALUE '09',

LAST(10)TYPE C,

END OF XSTRUC.

CSTRUC = XSTRUC.

"UNICODE ERROR!

The example above will cause a Unicode error.

Page: 58/63

ABAP Programming Standards 

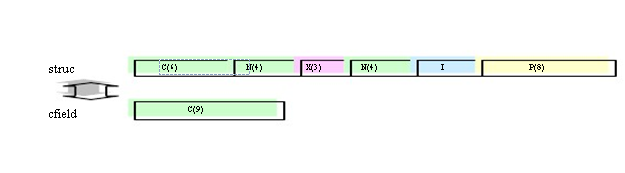
2.

MOVE between Non-Character-like Structures and Elementary Fields

To move between non-character-like structures and elementary fields

The elementary field must be of type character

* The elementary field must be no longer than the character prefix of the structure.



Example:

DATA: BEGIN OF REC\_PERSON\_DATA,

AGETYPE I,

NAME(20)TYPE C,

END OF REC\_PERSON\_DATA,

V\_WA(1000) TYPE C.

V\_WA = REC\_PERSON\_DATA.

"UNICODE ERROR!

The example above will cause a Unicode error.

8.1.5.2.4Similar Restrictions Like MOVE

1. Implicit Moves e.g. operations on internal tables

LOOP AT i\_tab INTO rec\_wa.

2.

Structure Comparison

between incompatible structures

between structures and elementary fields

Moving/Comparing Internal Tables

Similar rules based on line types of tables apply.

Database Operations

Select into and update from work areas and internal tables

SELECT \* FROM dbtab INTO rec\_wa ...

SELECT \* FROM dbtab INTO TABLE i\_tab ...

UPDATE dbtab FROM rec\_wa ...

3.

4.

...

8.1.5.2.5

1.

Access with Offset or Length

The following restrictions apply when accessing structures with offset or length







Structures must begin with character types

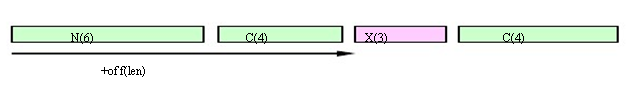
Offset and length are counted in characters

Access is only allowed within the character type prefix of the structures

Page: 59/63



ABAP Programming Standards



2. The following restrictions apply when using ASSIGN fld+off(len)

Access must not exceed field boundaries

If ASSIGN fails, the field-symbol is set to 'unassigned'

NEW ... RANGE addition allows the permissible boundaries to be expanded

8.2 Language

The ESAP implementation will span multiple countries and hence languages. Therefore any interaction with the user, either through screens or output, language translation should be possible. Therefore not use of literals, or hard coding for messages, labels. Text elements, like text symbols, selection texts should be used. Note: Oss note 585116 Translation after installation of WebAS620.

8.2.1 Text handling

INCLUDE files can't define their own Text Elements - any Text Elements to which they refer must be defined in the main program which invokes the INCLUDE file. Therefore, if it is possible that an INCLUDE file may be   
invoked from more than one main program, constant text that is used within the INCLUDE file should be   
defined with the CONSTANTS statement.

In cases other than INCLUDE files, constant text that is printed in a report can be stored as Text Symbols.

There are two ways that you can code references to these Text Symbols, either using TEXT-xxx or using

'...'(xxx). Here, xxx stands for a 3-digit number, and ... for the text of the Text Symbol.

The first form requires that you separately define a Text Symbol for number xxx. If xxx isn't a defined Text

Symbol, the output is empty. The second form improves the readability of the program. The text between the

single quotes should correspond to the text stored as the value of the Text Symbol. If it does not, the system uses the text stored for the Text Symbol. Exception: If there is no text saved under number xxx, the text   
between the single quotes is used.

Example: Text symbol number 001 has the text 'Please enter your name'. The following commands:

WRITE: / TEXT-001,  preferred option

/ 'Please enter your name'(001), / 'What is your name?'(001).

all have the same output: "Please enter your name". In the ABAP Editor, you can compare the texts used in the program with the texts stored as Text Symbols by choosing "Goto -> Text elements -> Text symbols", then   
"Utilities -> Adjust -> Text symbols", then selecting the "Text symbols defined repeatedly/differently in   
program" radio button and clicking on the "Edit" soft button.

The advantages to the '...'(xxx) form are readability and that the Text Symbol only needs to be maintained for multi-lingual clients (for those installations which use multiple languages). The advantage to the TEXT-xxx form is easier maintainability if TEXT-xxx is coded several times in the program and it needs to be changed.

8.3 Country specific requirements,

Again as this implementation programme spans across multiple countries, there could some country specific requirements, especially in the area of enhancements. No hard coding will be allowed in these cases;

i.e. If Country eq „GB‟

Do country specific logic

Page: 60/63



ABAP Programming Standards

endif.

Is not allowed.

Instead a bespoke table would be used to for this purpose.

Page: 61/63



ABAP Programming Standards

9 Quality Assurance

9.1 Adherence to Programming Standards

Any program object developed will be considered complete if it adheres to the programming standards, and also passes the Unit test. The details of the Unit test would be covered in the Test Strategy document.   
Completing a Checklist form as part of the development completion activity would check adherence to the   
Programming Standards. Most of the checks in the checklist would be automated using Code Inspector   
(transaction SCI); the idea is reduce manual work and thus manual errors as far as possible. A column in the   
checklist will specify which checks will be performed using Code inspector. In cases where performance of the development is critical to the functionality, like very high volume, or real-time response, premium checks using tools such as Performance analysis (transaction ST05) would be performed.

Checklist Summary:

Technical specification is up-to-date and all relevant sections are complete. Online documentation is maintained properly

Naming convention is followed

Code Indentation is complete

Performance standards and guidelines are followed Error Handling

Security

9.2 ABAP Unit Test

ABAP Units is tool, which enables the developer to test units of code, by writing test methods within the code.   
Since these tests are created within the program, they can be easily synchronised if there are any changes in   
the program. The ABAP Unit are written using ABAP objects, so no new skills are required for creating these   
tests. Also these tests are not transported to production system therefore do introduce any execution   
overheads.

By setting system parameter “ABAP/test\_generation“(in transaction RZ12), one switch on or off ABAP UNIT class compilation in a system. It will be generally switched off in production.

Page: 62/63



ABAP Programming Standards

10 Security and Authorizations

10.1 General

Transactions SE16, SM30, SM31 SA38 and SE38 are generally restricted in Production. All Reports and

Programs to be run directly by users should have an associated Report Transaction that can be run from an appropriate Report Tree.

Authorization Groups

For viewing and maintaining data in tables it is relevant to check the organisation level data of the user, for example which Sales Organisation they belong to. For Reports and Programs it is relevant to enter an   
appropriate Authority Group.

10.2 Reports and Programs

Reports and Programs must have an appropriate Authorization Group set in the Program Attributes. To check Authorization from ABAP, use the command Authority-Check.

10.3 Viewing Tables

To replace SE16 a new Parameter Transaction needs to be created per table to call SE16 so that the users who we can restrict which users can call the Transaction.

10.4 Maintaining Tables

A Parameter transaction should be created to call SM30/31. This will maintain a View of the actual table, that is restricted to one Country. The Authority Group will be in the Table Maintenance of the View.

10.5 Dialogue, Report and OO Transactions

With these Transaction types the appropriate Authorization Object should be checked with the appropriate values. For example, you could use ZTAB\_VKORG, to ensure that the transaction is used only by certain Sales Organisations (i.e. Countries).

10.6 ABAP Queries

An Authority Check should be coded using the command ‘Authority-Check’ or Function Module

‘Authority\_Check’ to ensure only users with the correct profile and/or from the correct Country can run the   
Query.

Page: 63/63