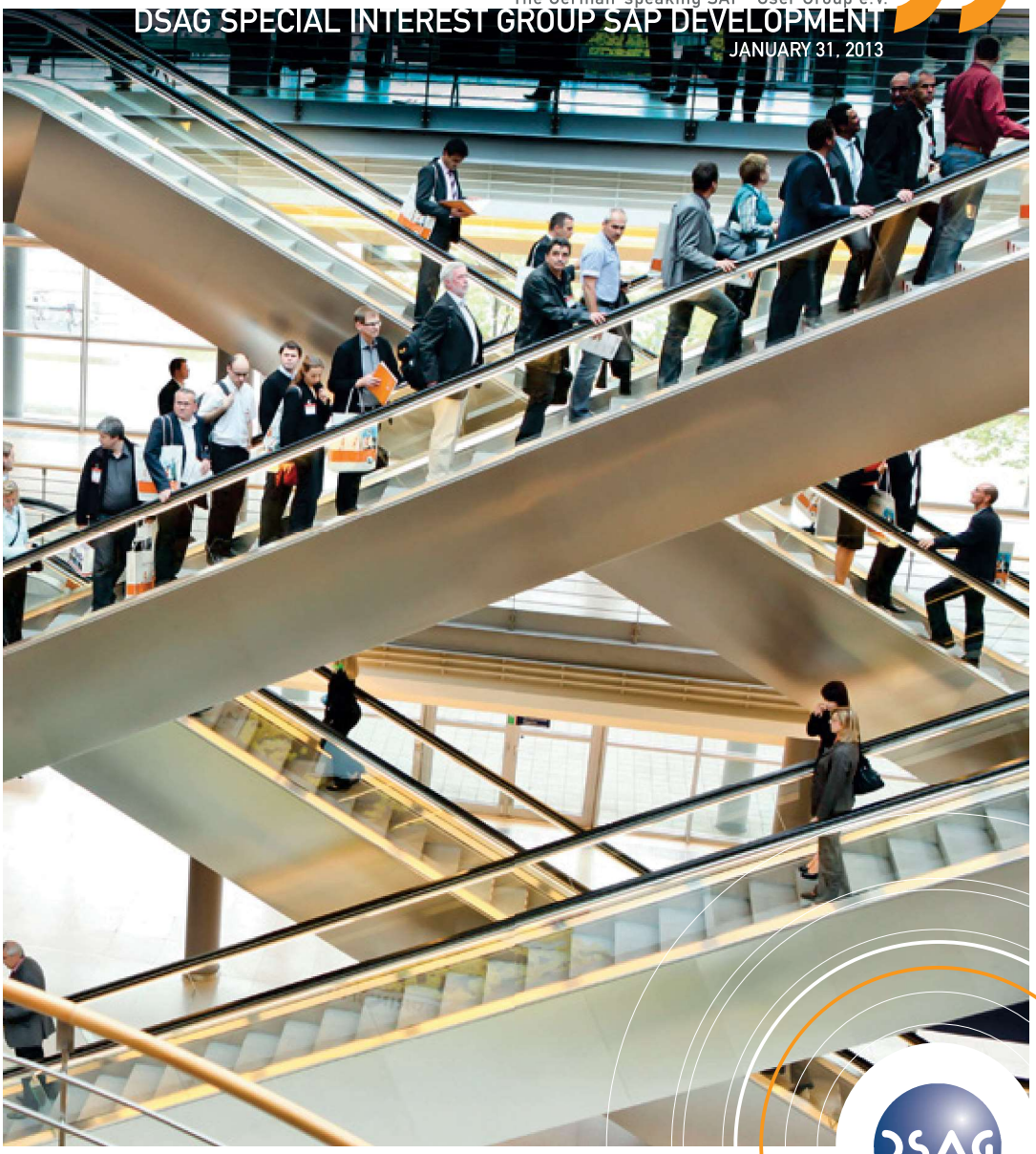
Best Practice Guidelines for Development

Practical tips on the ABAP Development

The German-speaking SAP® User Group e.V.

DSAG SPECIAL INTEREST GROUP SAP DEVELOPMENT

JANUARY 31, 2013



2

Best Practice Guidelines for Development

Practical tips on the ABAP Development

VERSION 0.11
AS OF: JANUARY 31, 2013

DSAG e. V.
Deutschsprachige SAP®-Anwendergruppe e. V.

AUTHORS

- Peter Lintner, Senior Consultant, Allgemeines Rechenzentrum GmbH
- Steffen Pietsch, Vice President, IBSolution GmbH
- Markus Theilen, IT Coordinator, EWE AG
- Jürgen Wachter, Process Coordinator Development, Comgroup GmbH
- Michael Werner, SAP Applications Consultant (Inhouse), LTS AG Andernach
- Andreas Wiegenstein, Managing Director and Chief Technology Officer (CTO), Virtual Forge GmbH

For more information on the authors, we refer you to page 57.

DEUTSCHSPRACHIGE SAP® ANWENDERGRUPPE E.V. (The German-speaking SAP® User Group)
Altrottstraße 34 a
69190 Walldorf
Germany

Phone: +49 (0) 6227 - 35809 58
Fax: +49 (0) 6227 - 35809 59
E-Mail: info@dsag.de
Homepage: www.dsag.de

DSAG
Deutschsprachige
SAP® Anwendergruppe

TABLE OF CONTENT

TABLE OF CONTENT

1 INTRODUCTION

As standard software, SAP software is characterized by a high degree of flexibility and expandability. There are customer-specific adaptations and enhancements in almost all companies using SAP software. Thus, SAP software is subject to a process of continuous adjustment and extension to changing customer needs both on the manufacturer and on the customer side.

The high degree of flexibility and expandability of SAP software brings both advantages and disadvantages. The software can be adapted optimally to customer specific requirements which may bring added value. At the same time this expandability runs the risk of resulting in customer specific development that is complex, error prone and requires ongoing (and possibly burdensome) maintenance.

The aim of this document is to provide practical tips, thoughts on maintainability and efficient creation of customer-specific developments.

1.1 MOTIVATION

The work of the Deutschsprachigen SAP-Anwendergruppe e.V. (DSAG for short, in English – German-speaking SAP user group) is based upon three pillars – knowledge, influence, and networking. The present document was initiated by the members of the DSAG working group for the SAP NetWeaver Development and references the first pillar, i.e. the knowledge for users and partners. As a team of authors it is our intent to provide information regarding the topic "development" that exists and is distributed currently amongst our member companies and to make that knowlege available to the other DSAG members in the form of a compact document. It is our plan that this document "lives" and that it is subject to an ongoing improvement gained from your valuable experiences. We appreciate your feedback (preferably by e-mail to handlungsempfehlung@dsag.de)!

1.2 POSITIONING

SAP and a number of specialized publishing houses already provide very good publications on application development and extension of the SAP platform. Especially with the book "ABAP-Programmrichtlinien (programming guidelines for ABAP)", SAP Press 2009, authors from SAP have already taken an important step towards recommendations going beyond the mere description of the ABAP language and the associated tools.

The added value of this document lies in the summary of best practices, practical tips, and proven system(s) of rules from the user companies. These guidelines shall provide you – as users, developers, development managers, project managers and IT managers – with recommendations and assistance so that you "do not have to re-invent the wheel" that enable you to build upon experience gained by others. The recommendations presented by these guidelines do not claim to be complete nor to be generalized, but they do represent a selection of practical tips.

As a team of authors we have endeavored to find the right balance between general and detailed knowledge. Therefore, in order to avoid the repetition of topics that have been discussed in-depth, we refer you to other sources at the appropriate places. The first edition of these guidelines is focused upon the field of ABAP development. With the corresponding feedback and your active support, the focus can also be directed to the JAVA development and other topics.

2 PROGRAMMING GUIDELINES

This chapter describes proven and recommended programming guidelines for applications that are created by means of the ABAP programming language. A description is given of how a clearly readable and comprehensible ABAP code can be developed with standard SAP tools and discipline. This facilitates the maintenance of the code and/or enables an efficient cooperation with different internal and external persons in the (further) development and maintenance of a program.

2.1 NAME CONVENTION

Name conventions describe the uniform and binding guidelines for the naming of software objects (e.g. classes, function modules) and/or for naming the object in the source code (e.g. variables).

We strongly recommend specifying a name convention as a directive for developments in the SAP system. The purpose of using a standardized name convention is to considerably increase the maintainability of customer-specific adaptations and expansions. As a consequence, this leads to lower maintenance requirements and/or costs and to a faster troubleshooting in case of an error.

To make the new employees familiar with the general rules and the company-specific requirements, the explicitly formulated name convention should be part of the internal training. Moreover, it has proved positive that this name convention is made the subject-matter of the contract for external developers and partner companies. Automated checks ensure that the requirements are met. (cf. Chapter 7).

BEST PRACTICE: You find a template of an exemplary name convention in the Appendix.

2.2 NAMESPACE

The separation between customer objects and SAP objects can be carried out via the prefixes Y or Z as well as via an own namespace. The syntax is as follows:

Z...
Y...
/<customer-specific namespace>/...

The customer-specific namespace can be registered with SAP. After SAP's confirmation, the namespace can be identified worldwide and it is registered for a use by the respective company. This process supports the conflict-free assignment of names for software objects.

The advantage of the customer-specific namespace is the guaranteed absence of overlaps when importing external projects into the own SAP system (e.g. when using external applications that are imported by a transport request) and when combining the SAP systems within the scope of a post-merger integration. With the reservation of the namespace, it is assured that a software object with the same prefix cannot be created on an external, e.g. non-registered system.

The disadvantage of using the customer-specific namespace is that at a continuous use of the prefix, several characters are already "consumed". This can lead to difficulties especially with objects that provide only a few characters for a naming. In addition, the use of namespaces is not supported by all of object types, e.g. authorization objects.

BEST PRACTICE: We highly recommend using a customer-specific namespace.

OTHER SOURCES:
1. http://help.sap.com (setting up a namespace)
2. Best-Built Applications: http://scn.sap.com/community/best-built-applications

2.3 A STANDARDIZED AND READABLE SOURCE CODE: PRETTY PRINTER

A clear and readable code facilitates the (re-)familiarization with the source code for each developer. The easiest and fastest way to make and maintain the code readable is to use the Pretty Printer from the ABAP development environment. By the push of only one button, the selected source code is formatted in a standardized way. It offers various options that can be configured via the settings of the workbench. An indented presentation already makes the source code more readable. It is recommended to capitalize the keywords. That way, the source code can easily be understood even in a printed form and without syntax coloring. The Pretty Printer provides an easy way to create a standardized source code – despite different developers.

For an improved readability of the source code, we recommend refraining from several instructions in a line of code.

We recommend deactivating the option "Insert standard comments", as the generated comments are not automatically adapted to modifications and include redundant information.

BEST PRACTICE: We recommend using the Pretty Printer and defining the settings as a uniform standard.

Modularization

Programs that are not divided into logical units of work cannot easily be read as a result and thus, they cannot be maintained and extended.

A modularization unit (form routine, method, and function module) has to summarize instructions that logically belong together. It must, however, be borne in mind that the individual units do not cover trivial functions. Modularization units with only a few instructions have to be avoided.

Despite the complexity of the task, the modularization is designated to clearly arrange the program code. Moreover, program sections with the same logic have to be avoided. For a practical implementation, it can help to divide the first lines of code into logical blocks by means of comments before starting the programming and only then to finalize the programming.

Wherever it is possible and useful, it is recommended to shift from the procedural programming model to the object-oriented programming for a future-proof development and encapsulation of the objects. Especially new projects should only be developed by an object-oriented programming.

A cleanup of the language and a standardization of the constructs took place during the implementation of ABAP objects. Thus, the use of ABAP objects leads to an increased maintainability.

2 PROGRAMMING GUIDELINES

Separation of presentation and application logic

A separation of presentation and application logic should always take place in all programs. In this way, results and functions of the application logic are displayed to the user by means of different UIs (User Interfaces) and they are made available to other systems via a standardized interface. This statement applies to all the current UI technologies; however, the degree of support and/or the compliance of this logical separation can be different. A separation between model and UI logic is already provided by the framework for a WebDynpro ABAP realization. The separation is not supported in the same way for classical dynpros and BSPs; but basically the separation can and should be realized in these environments. Contrary to the WebDynpro, there is no technical examination where the corresponding checks are realized by the Code Inspector.

Rules for plausibility checks are a typical example for a clear separation of application logic and UI. If the plausibility check for entries is developed in a certain UI technology, these checks have to be re-developed when switching over to another UI technology. To avoid this, the functions for a check of entries or parameters have to be created and maintained regardless of the UI used.

Internationalization

Language-dependent texts in programs must not be "hard coded", but they must be stored in text elements (program texts, class texts, Online-Text-Repository [OTR]), standard texts, or message classes. Since all in-house developments are qualified to be used worldwide, they should be translated into the most important languages.

Moreover, language-dependent (customizable) texts have to be filed in own text tables. This text table has the same key attributes as the actual customizing table. In addition, the first key attribute following the client field has to be the language attribute (data element SPRSL or SPRAS). Furthermore, the foreign key dependency has to be designated as one of the text table.

BEST PRACTICE: We recommend using the Code Inspector for the search of texts that cannot be translated.

BEST PRACTICE: To make subsequent translations easier, the length of the field labels and text elements should be selected as long as possible. As a thumb of rule for the length of text elements, 1,5 times the length of the native description has proven its worth.

Dynamic programming

In the "traditional", static development, the development objects and the source code are defined at design time and they are stored statically in the SAP system. The predetermined program code is executed during runtime. In contrast, the dynamic programming makes the source code flexible. The dynamic programming is illustrated by the following example:

The name of an ABAP class to be activated is not stored statically in the program code, but the class instance whose name is defined by the contents of a variable is accessed during runtime. This name can vary for ex. due to user entries. The increased flexibility is the advantage of this method. The considerably growing complexity and, in particular, the security risks associated with it are the disadvantage.

Advantage:

- Significant increase of flexibility
  - Example 1: Own structure of user exits

    The basic structure for an "user exit" is predetermined by the static definition of an abstract class, including the method signature. Afterwards, several specific implementations of this abstract class can be created. Within source codes, the name of the specific class implementation to be used is read for ex. from a customizing table and then it is activated. Thus, different implementation variants can be activated/deactivated in customizing.

  - Example 2: Dynamic WHERE-condition

    The WHERE-condition for a database operation, e.g. SELECT, is created with a string variable during runtime. Complicated CASE queries that, depending on the entries, execute different OSQL commands can be avoided that way.

Disadvantage:

- Through the use of dynamic calls, the where-used list within the ABAP development environment is lost. Then, it will be difficult to change the activation targets. A developer who changes for example the passed parameters of a function module being called dynamically by a program will not notice the usage by calling the where-used list.

- As a rule, a syntactical check is not possible at design time during the dynamical programming; if the variable contents (e.g. wrong compounding within the dynamic WHERE condition, wrong name of a class) are not correctly assigned, there will be an unplanned termination of the program (short dump)

- A dynamic programming represents high security risks, especially where the dynamic contents can be influenced by unprotected accesses (e.g. if the name of a class to be activated/a function module or a WHERE condition can be influenced by user entries. Key word: code injection).

BEST PRACTICE: The dynamic programming should only be used in a very dosed and controlled manner. A program code containing dynamic parts should be controlled and documented in accordance with the principle of dual control, as it represents a potential security risk.

The topic of dynamic programming – also with regard to security – is dealt with in chapter 5.2.

2 PROGRAMMING GUIDELINES

Auditability of the ABAP Code

It must be possible at any time to check the ABAP code written by the user for the existence of any defects by means of manual examinations or static code analysis tools. Therefore, all methods to make the ABAP coding invisible are not permitted, as they could impede such examinations or they even could be used in a targeted way to smuggle backdoors in a system. Even a disguised code can no longer be examined by the debugger. Techniques for hiding codes are explicitly set aside in this context.

BEST PRACTICE: Please do not use any techniques for hiding your source code especially for the development in the own company.

2.4 OBSOLETE STATEMENTS

Although SAP represents a strict downward compatibility, it has to be considered that the use of obsolete statements, such as headers in internal tables can cause problems if the code has to be transferred to classes. Here, it should be noted that there are always modern alternatives for obsolete language elements. Out of habit, there is little reason to use them. Therefore, they should be avoided.

BEST PRACTICE: For the detection of obsolete statements, we recommend using a static code analysis tool at regular intervals. The Code Inspector from the SAP tools and/or the performance of a syntax check can be used for this purpose.

Furthermore, there are very good analysis tools from third-party providers.

2.5 SYNTAX CHECK AND CODE INSPECTOR

The syntax check and the Code Inspector allow the verification of the program code at design time.

When the transports are released, the Code Inspector can be switched on globally via SE03 for the detection of errors. That way, the number of transports can be reduced, as the release can still be cancelled at the detection of errors. Then, the correction of errors can be included in the existing order and no new transport must be created.

BEST PRACTICE: In SAP standard, the Code Inspector is only carried out at the release of a transport request. However, a check by the Code Inspector is already recommended for the release of the respective transport task.

OTHER SOURCES:
The procedure how to implement the necessary BAdI is described in the technical publication "Praxishandbuch SAP Code Inspector" (SAP Press).

2.6 FIXED ENCODING: NO "MAGIC NUMBERS"

The fixed (hard coding) encoding of texts, numbers, user names, organizational units, dates, etc. should be explicitly avoided in the source text.

The direct use of hard coded values can appear as a seemingly fast procedure during development. However, with reference to the whole life cycle of the application, it leads to considerably increased costs. The maintainability and testability of the application are significantly complicated in the long run.

BEST PRACTICE: We recommend using constants that are defined by a central position. For ex. attributes of global classes and/or interfaces are qualified for a definition of constants

BEST PRACTICE: The customer-specific customizing tables can be used as an alternative to a hard coding in the source text. The values, such as numbers, organizational units, etc. are stored as a configuration in these tables and they are read as a variable when the program is started. Afterwards, this variable is only used for further processing.

For multiple uses, the above-mentioned procedures increase the consistency and efficiency of the analysis in the case of an error and/or at regular maintenance activities. We expressly recommend that a central position provides the constants with comments and the values with a meaning.

To help new employees determine the constants, it may be advisable either to document them in a searchable way outside the system or to make a search tool available (see "Other sources") that is searching for constants within the system.

2.7 TIPS ON HOW TO WORK WITH TRANSPORTS

If several developers make changes in a development object, this may lead to problems when the transport requests are not transported to the productive systems in the correct order or when other objects from different transport requests are missing.

BEST PRACTICE: To avoid this, it should be worked with a transport of copies. In doing so, the changed development objects are blocked by the actual transport request in the development system and copies of the objects are only transported to the quality assurance system via a transport of copies.

A developer will immediately notice that another developer is already working on this object and so he can coordinate his activities with him. The blocked original transport is only transported to the productive system when the "project" is completed.

2 PROGRAMMING GUIDELINES

2.8 AUTHORIZATION CHECK IN THE SOURCE CODE

The authorization objects that are necessary for an access to data as well as to its presentation have to be checked. However, when using standard objects, the corresponding SAP standard authorization objects have to be checked (this makes the maintenance of the necessary roles easier). As a general rule, SAP standard authorization objects are not used to check customer-specific data objects. For this purpose, customer-specific authorization objects can be implemented and checked.

For more information, we refer you to chapter 5.1.1

2.9 PROGRAMMING MODEL: OBJECT-ORIENTED VS. PROCEDURAL

In the meantime, the procedural development in ABAP has been classified as obsolete by SAP. In particular, the commands FORM...ENDFORM und PERFORM are classified as obsolete statements. Over the years, the procedural development has proved to be very confusing, complex, and error-prone – even with regard to global variables and includes.

The object-oriented development has been designed to combine all tasks that logically belong together in uniform objects. Among other things, the reusability of the codes is increased that way. Without affecting the basic functions (open/close principle), even other developers can easily extend and change such objects for their purposes. On the other hand, central functions or individual variables can be protected in a targeted way from an unwanted read or write access of the programs to be called

BEST PRACTICE: We recommend working only with classes and methods, if possible, and no longer using any FORMs for new developments.

OTHER SOURCES:

1. Horst Keller and Gerd Kluger, Not Yet Using ABAP Objects? Eight Reasons Why Every ABAP Developer Should Give It a Second Look, Sap Professional Journal
2. Horst Keller and Gerd Kluger, NetWeaver Development Tools ABAP, SAP AG
3. Bertrand Meyer, Objektorientierte Softwareentwicklung, Hanser 1990, ISBN 3-446-15773-5
4. ConSea project: search for constants on the SAP Code Exchange:

   https://cw.sdn.sap.com/cw/groups/consea

2.10 OTHER SOURCES (PROGRAMMING GUIDELINES/ABAP)

1. SAP Dokumentation SAP NetWeaver AS ABAP Release 731
   http://help.sap.com/abapdocu_731/de/index.htm
   This documentation includes a special chapter on the Programming Guidelines.

3 PERFORMANCE

To avoid performance issues from the start the following rules should be taken into account in the course of routine ABAP-development. As is the case for other aspects of software development, performance will be improved if you know what should actually be done. If the reason for a piece of code is not apparent, this needs to be clarified first.

3.1 PRINCIPLE OF AVOIDANCE

"The most secure, fastest, most precise, cheapest, easiest to maintain, most reliable and easiest to document pieces of a computer system are those that aren't there." (Gorden Bell)

BEST PRACTICE: avoid any unnecessary coding.

In order to follow this meme you should evaluate closely which code is really needed in production and delete all test- and prototyping programs in the QA-system at the latest point in the transport process to a production system.

With production code you should also always work from the position to not do more than the task requires. A clear example of this is the rule that you should avoid 'Select *' which is an easy way to select all columns of a table even though the subsequent processing may only require a couple columns.

Hint: avoiding 'Select *' improves the robustness of programs. E.g. when one reads data with the help of 'Select *' into a customer-specific structure from a standard table any subsequent table enhancement by SAP may lead to errors. Reading only the necessary columns helps to avoid this issue.

3.2 USING EXISTING TOOLS

The tools available in the SAP-system support the creation of performance-optimized applications as well as the analysis of performance issues. These tools should be used early on and throughout the life of the software. The table below gives an overview of the general tools:

Development	Description
Code Inspector	Static code analysis and checks
SE30/SAT	Runtime traces
ST05	Traces for SQL, RFC and enqueues
DB05	Analysis of table with respect to index fields

3 PERFORMANCE

Development	Description
SM50/SM66	Process overview/application server
Debugger	Step by step program execution
Memory Inspector	Comparison and analysis of core images to check memory usage and unreleased heap objects
ST10	Table call statistics to verify table buffering

Post Mortem	Beschreibung
ST22	Analysis of runtime errors (e.g. memory issues)
STAD	Workload analysis
ST04	DB performance overview

BEST PRACTICE: start any performance analysis with a trace via SE30/SAT and concentrate on the biggest time hog. If most of the time is spent in the ABAP part, keep analysing with SE30/SAT. If most of the time is spent with database calls then use the SQL-trace in transaction ST05.

3.3 PERFORMANCE OPTIMIZATION ONLY AT CRITICAL AND RELEVANT PLACES

Obviously, you should always avoid creating software with inherent performance issues. However, if the performance needs to be optimized you should limit your focus on those areas of the software where you know via measurements that they are the root cause for either the long runtime or increased memory usage. The 80/20 rule also applies to performance issues and it is therefore important to use the usually scarce resources on improving the 20 % of the system which is responsible for 80 % of the runtime/memory usage. To find these places, sensible usage of the listed tools is essential.

BEST PRACTICE: our recommendation is to start the search for performance bottlenecks with runtime analysis SE30/SAT with full aggregation. This should clarify whether the runtime is a result of the interaction with the database or whether it is due to the processing of the data loaded into the main memory.

It is important to use representative and realistic data for the processing to avoid being led astray by rare processing patterns. If more than half of the runtime is being spent with database processing, you should do a closer analysis of the SQL-commands via transaction ST05. Use SE30/SAT for more in-depth analysis if more runtime is being spent during ABAP processing. Narrow down the aggregation level from "full" to "none" to get a more precise feedback about critical processing areas. Compare and document your results after each optimizing step.

3.4 DATA MODEL AND DATA ACCESS

3.4.1 Data Model and Indices

Setting up the data model is the basis for performance optimized applications. A pragmatically normalized data model is able to work efficiently with the data and indices. The normalizing rules are applied independently of the programming language ABAP. Our recommendations for utilizing table indices are listed below:

- For tables being accessed very often in update mode, there should not be more than five indices. The maintenance costs related to data changes increase with each index.
- The upper limit for number of fields in an index should be five.
- You should only include selective fields in an index and they should be listed by decreasing selectivity.
- You should not have any overlaps on the field level between two indices for a table.
- For client-dependent tables, the field "Client" should be included as the first field. This applies especially to large tables with more than 1,000 entries. Even though this field is not especially selective and therefore runs counter to the third rule, it is included and queried by the SAP system for each query. Not having this field in the index can have negative effects for tables with widely different number of entries in the clients.

3.4.2 General Framework for Database Access

The following questions should be asked and answered with regards to database access:

- Are suitable indices available?
  - See previous section (3.4.1)
- Are select commands ignoring the table buffer?

You can use transaction ST10 to check the number of selects on buffered tables. The combined SQL- and buffer-trace of transaction ST05 can be used to determine selects which do not use the table buffers. In addition to this, the code inspector can be used to highlight many of these statements beforehand.

- Is it possible to use a database index to sort the data with ORDER BY?

In the simplest scenario the addition PRIMARY KEY can be used if the sort order should be done according to those fields. If other sort orders are needed the creation of an index with the relevant fields in the proper sequence can help. The hints regarding the number of indices per table should however be considered. If the addition ORDER BY is rarely used it is usually not necessary to create a separate index for it. The sorting should then happen in the ABAP code if the amount of data to be sorted allows it.

Example: a table contains a document date and you want to get the latest 'n' documents sorted by that date. In this scenario, an index for this field would most likely be helpful, because the field is included in the WHERE-clause and the addition ORDER BY can then at the same time profit from the existing index.

3.4.3 Database Access

The amount of data selected from the database and provided to the application level in general should be as small as possible. Below, you will find some tips how this may be achieved.

BEST PRACTICE:

> Reduction of selected columns:

Avoid "*-selections" and instead list the column names needed in the selection. Especially expensive are unnecessary retrievals of columns with type STRING. The result table columns should be synchronous with the selection structure for optimized results. If you have more fields in the result table but if the fields to be retrieved have been given identical names to the ones used for the table, you can use the addition CORRESPONDING FIELDS. This does not cause additional runtime. Simultaneously, the robustness of the code is improved.

> Optimized query:

If at all possible, try to use all fields of an existing index for the database query. If this is not possible, try to at least take the first index fields into account. This will restrict the sequential search to as few data records as possible.

> Reduction of line items

Use the WHERE-clause to restrict the selection and to minimize the amount of data returned to the ABAP system. Use SELECT SINGLE/UP TO n ROWS, whenever you only need some lines.

> Existency checks

Do not use COUNT(*) to find out whether there are records for specific selection criteria. Use SELECT SINGLE <fieldname> instead, with a field included in the index accessed for the selection. This avoids unnecessary table access.

> Aggregate

Aggregates like MIN or MAX are always resolved on the database server and the table buffering is therefore circumvented. This can potentially cause a high load on the DB system if it is accessed from several installations and application servers. Developers should therefore check how large the data volume for the aggregate will be and if it could make sense – i.e., if it is not too large – to first load the data into an internal table and then do the aggregation from there. We would like to point out, however, that as far as it is known today, HANA based database queries do have their strengths especially for aggregations and that their usage is therefore explicitly encouraged for HANA.

Example: to calculate the average amount of a large number (>100,000) of orders it makes sense to have the DB-system determine this aggregate. Then it will pass only one or a few items instead of hundreds of thousands back to the application server for calculating the average amount in ABAP.

This is applicable especially if this calculation is done rarely (e.g. just once per day). If, on the other hand, you often need to determine the sum of the order position of a single order and on all available application servers, doing the calculation in ABAP is usually the better option.

> Updates

The command UPDATE SET makes it possible to restrict the list of fields to be updated (instead of updating the complete record). This command should be preferred if possible.

> Number of DB-access executions

Each execution of an open SQL-statement comes with a certain overhead (parsing, checking against the statement buffer in the DBMS etc.). Each command should therefore retrieve as much data as possible at once. E.g. If you need data of 50 orders, you should not get them in 50 individual selects but you should retrieve them via one statement which supports the so called array-fetch. These commands can be identified by the additions INTO TABLE for SELECTs or FROM TABLE for UPDATE statements. Avoid Open SQL-statements within loops at all cost! With these types of constructs you will have the overhead for the statement at each loop iteration.

Do not use MODIFY <DB-table>! Within an application it should be clear if data records were created or if existing ones were updated. In addition, the statement is extremely critical from a performance perspective. Even with the addition FROM TABLE, the database is accessed once for each line item in the internal table. For each of these, an UPDATE is tried first and if this is not successful, an INSERT is done. If you have many new data records to insert, this will not happen with n database access but it will be 2n where n = the number of line items in the internal table.

> VIEWS/JOINS

Nested SELECTs and SELECT-commands in loops should be avoided. As an alternative, make use of VIEWS, JOINS or the addition FOR ALL ENTRIES. Please keep the following in mind with FOR ALL ENTRIES:

a. If the internal table referenced with FOR ALL ENTRIES is empty, all items will be loaded.

b. If the internal table contains duplicate entries, it is possible that the related data records will be loaded twice from the database. It therefore makes sense to get rid of duplicates via DELETE ADJACENT DUPLICATES.

3.5 INTERNAL TABLES AND REFERENCES

Internal tables are a central construct within the application development with ABAP. In addition to database access they are another prominent source of performance issues. If the data volume is small, the type of table or key does not play a big role. Once large volumes are being processed – which is what often happens once the code has been transported into the consolidation system – you can run into large runtime increases even in places in the code deemed to be uncritical.

3 PERFORMANCE

BEST PRACTICE: the following hints should be taken into account to improve the performance of applications:

- Chose the table type which best fits the intended usage:
  - Standard tables are suitable for data which are rarely or not at all searched for specific criteria. If no searches are needed, then it is not worth the costs to create and to keep current the additional key-structures needed for the other table types. Depending upon the number of search requests and the table width and if the amount of data are very small (<100 lines) it may be feasible to forego an explicit key altogether.

  - Sorted tables are suitable if the data often need to be searched via (partial) keys, but if it cannot be guaranteed that the key fields are unambiguous. READs which only use some of the first key fields should be done only with this table type.

  - Hashed tables are perfectly suited to search for unambiguous keys in dictionary like constructs. If the unambiguousness of the entries regarding the key fields can be guaranteed, and if the search term always uses the complete key (all the fields of the key are checked against the corresponding value) this table type is usually the best.

- If a table of type SORTED or HASHED is accessed, this should always be done with a suitable (partial) key.

  This means to use WITH TABLE KEY for READ TABLE and to query as many of the key fields – in their proper sequence – as possible with "=" in a LOOP AT WHERE construct. If this is done, the internally built key-structures will be used to find the corresponding entries as quickly as possible.

- Starting with AS ABAP 7.02 it is possible to define additional secondary keys for internal tables which are rarely changed but which are accessed with more than one access pattern. These keys can have a different type than the one used as the primary key (sorted, hashed). You still need the primary key which gets defined and used as before.

  As an example you can define an additional key with type sorted for a hashed table with an unambiguous primary key. This additional key makes it possible to access data in the table from another perspective (ambiguous, partial key possible) without the need to load the data a second time into the main memory. Then there would be no need to manually ensure consistency between the two tables.

- Similar to DB-access there are single and mass operations for internal tables.

  Whenever possible, the mass operations should be used because they are performance optimized as compared with multiple single operations. E.g., appending lines from a partial results table to an overall results table should be done with the command pattern APPEND LINES OF... TO instead of doing the same thing with a LOOP AT and single APPEND TO.

- When using the SORT command, always provide the needed sort fields. This improves the legibility of the code and standard table types more often than not do not have a table key defined. Without such a key, the complete table line is used as the key and all the fields of the

table are checked during sorting which leads to a considerable loss of performance. If a table needs to be sorted by user and date, use the command SORT table BY USER DATE even if the table structure starts with these fields and even if the sequence of the results regarding the requested fields is the same.

- Before using the command DELETE ADJACENT DUPLICATES you should always ensure that the table has been sorted by the same fields so that duplicate entries are actually eliminated.

  As the command indicates, only adjacent table rows are compared. Similar to the SORT command you should always provide the fields which are to be considered. Otherwise, the complete row will be compared field by field even if only two fields are needed from a process perspective.

- If you are only interested in the existence but not the content of a table row and if it is not needed for subsequent processing, always use READ TABLE TRANSPORTING NO FIELDS.

3.5.1 Field Symbols

Field symbols make it possible to reference existing data, e.g. .rows in internal tables. Working with references is distinctly faster than copying the data. Therefore, you should use field symbols whenever possible. There is only a miniscule runtime advantage and only for very small tables between the costs to copy the data via INTO <WA> and using field symbols. Apart from this, field symbols are always faster, especially if table content needs to be changed. When using field symbols you have to be aware, however, that each change of a field symbol's value, also changes the value in the referenced data element.

BEST PRACTICE: routinely make use of field symbols when accessing internal tables.

3.5.2 Passing Parameters

Passing values via parameters should be done only where it is technically mandated (e.g. RFC function modules, returning parameters for functional methods). This will avoid unnecessary copy costs for passing the parameters. This is relevant especially for parameters with deep data types like internal tables or strings. If there are no restricting technical reasons, parameters always should be passed by reference.

In addition, as few parameters as possible should be defined. Avoid optional parameters completely.

BEST PRACTICE: use as few parameters as possible which are passed by reference. Make use of the passing by value only where it is mandated technically.

3.6 ADDITIONAL RESOURCES

- SAP course BC490 offers a good grounding in the performance optimization for ABAP

- Siegfried Boes, "Performance-Optimierung von ABAP-Programmen", dpunkt Verlag 2009, ISBN 3898646157

4 ROBUSTNESS

In this chapter we describe measures developers need to take into account in order to write robust ABAP programs. Before we proceed, we first need to define the term robustness in the context of ABAP programs.

We define the robustness of a program as the ability to run and produce correct results even under unfavorable circumstance. In particular, a robust program needs to recognize and handle errors in order to preserve the desired functionality.

4.1 ERROR HANDLING

The ABAP runtime environment provides different error handling mechanisms. The following sections describe each of them as well as the related best practices for their usage.

4.1.1 Checking of SY(ST)-SUBRC

N.B. SY-SUBRC is an alias of SYST-SUBRC. Henceforth for clarity and simplicity only SY-SUBRC will be referenced since that is what ABAP code generally uses during checking in error handling.

When executing particular ABAP statements the SAP kernel sets the value of the global variable SY-SUBRC. Generally a SY-SUBRC value of zero indicates a successful execution of a statement.

The statements CALL FUNCTION and CALL METHOD do not automatically set the variable SY-SUBRC. The variable SY-SUBRC is only set when the following occurs. First, the called module needs to use non-class-based exceptions. Second, a value not equal to zero needs to be assigned to the defined exceptions in the EXCEPTIONS section of the statement.

BEST PRACTICE: always assign a value to the special exception OTHERS. This assures that the variable SY-SUBRC is set even if the list of exceptions of the called module changes. Especially when calling RFC function modules a value should be assigned to the exception OTHERS. The reason is that RFC function modules might raise special RFC-related exceptions.

The value of the global variable always needs to be checked immediately after the statement that sets SY-SUBRC. The reason is that any subsequent statement might change the value of SY-SUBRC.

4.1.2 The MESSAGE Statement

The MESSAGE statement is used to output status or error messages. The type of a message (status, information, error, exit) and the execution mode of the program (batch or dialog) defines the behavior of the MESSAGE statement.

BEST PRACTICE: avoid using the MESSAGE statement in modules without direct user interaction and in modules used in defined user dialog layers. The MESSAGE statement in conjunction with certain message types and execution modes might cause explicit COMMITs in the context of dynpros or the termination of a connection in the context of RFC function calls. In the core layers of applications, only class-based exceptions should be used to indicate errors.

4.1.3 Class-Based Exceptions

With ABAP 00 class-based exceptions have been introduced into the ABAP language. These exceptions use the throw-catch-paradigm for handling exceptions. A module indicates an error by "throwing" an exception on the call stack. If the caller of the module does not catch (i.e. handle) the exception it is propagated up through the call stack until it is handled. If the exception is not handled the execution terminates with a short dump.

When using class-based exceptions it is important to catch all exceptions that can be handled at the point they occur. Exceptions that cannot be handled need to be declared in methods and function modules where they might throw such an exception. Furthermore, exceptions should be handled in a consistent manner within an application. I.e. for the same error situation the same error handling or exception should be used.

The handling of class-based exceptions is performed using the TRY...CATCH...ENDTRY statement. All class-based exceptions of all modules of a program need to be handled using a TRY...CATCH...-ENDTRY statement in order to assure the robustness of an application.

BEST PRACTICE: empty CATCH clauses should not be used.

Using an empty CATCH statement makes the handling of an exception in the calling module impossible. Instead of using an empty CATCH statement the exception should be propagated.

BEST PRACTICE: the base class CX_ROOT should not be used in the CATCH clause of a TRY...CATCH...ENDTRY statement.

In the CATCH clause of a TRY...CATCH...ENDTRY statement the exception class defines the exception that should be handled. An example is the exception CX_SY_ZERODIVIDE to handle a division by zero error. Catching the base class CX_ROOT should only be done if the exception handling cannot handle all possibly unknown errors.

BEST PRACTICE: carefully choose the type of you own exceptions classes by inhering from the predefined base classes (CX_STATIC_CHECK, CX_DYNAMIC_CHECK, CX_NO_CHECK).

When inheriting CX_STATIC_CHECK each user of the exception class is forced by the syntax check to decide between the following possibilities. The first option it to handle the exception directly, the second to add the exception to the declaration and propagate it to the calling module and the third to encapsulate it into an own exception. Each of these possibilities requires development efforts and code changes that might eliminate the advantages of class-based exceptions (e.g. propagation of unhandled exception). Therefore, CX_DYNAMIC_CHECK generally should be used as a base class when introducing class-based exceptions. CX_STATIC_CHECK should only be used to consciously force a caller to deal with an exception.

The exception class CX_NO_CHECK is useful for exceptions that cannot be handled appropriately by a caller. Example of such exceptions would be the failure of a secondary database connection or other events that cannot be corrected in the program code.

4.1.4 Exceptions That Cannot Be Handled

Some exceptions cannot be handled in ABAP code and therefore inevitably lead to an aborting of the application and a short dump. In certain cases it is possible to check preconditions of the statement which raises an exception that cannot be handled prior to its execution.

An example is the OPEN DATASET statement. It causes a short dump if the user does not have sufficient authorizations to open a file. To prevent this, the user authorization needs to be checked (via the function module AUTHORITY_CHECK_DATASET) prior to calling OPEN DATASET.

4.2 CORRECT IMPLEMENTATION OF DATABASE UPDATES

4.2.1 Lock Objects

To prevent data inconsistencies it is necessary to lock business objects prior to changing them. Related business objects should only be changed if all related entities have successfully been locked. Only in this way can a conflicting database update can be prevented.

SAP locks are valid across several database LUWs (Logical Unit of Work). Therefore, consistent database changes can be performed across a business object.

The scope of a SAP lock should be chosen as specific as possible in order to only lock the relevant business objects within an LUW. Furthermore, locks should be retained for as short a time as possible but as long as necessary. To lock SAP standard business objects at the database level the respective SAP standard lock objects should be used. The reason is that these are also used in the standard transactions and therefore provide a consistent behavior.

For custom developments lock objects the related lock function modules should be implemented.

After an update has been performed the lock of a business object should be released again by calling the appropriate dequeue function module. In this context special care should be given to the scope parameter if update function modules are used.

FURTHER READING:

1. http://help.sap.com/saphelp_NW70/helpdata/de/7b/f9813712f7434be10000009b38f8cf/frameset.htm

4.2.2 Update Concept

The SAP update concept is the central technology to bundle database updates in a single database LUW and therefore for the definition of SAP LUWs in a SAP transaction. Database updates should not be performed using the statements of the data manipulation language (e.g. insert, update, delete and modify) within an application. Instead they should be performed using the update concept.

The update concept provides the following update techniques:

- asynchronous update
- asynchronous update in steps
- synchronous update.

The update is performed in a separate update function module. These function modules need to be marked as update function modules in their attributes. Update function modules are called with the addition "IN UPDATE TASK". Furthermore, some restrictions apply when developing update function modules. A program can send an update request using "COMMIT WORK". This statement executes all previously called update function modules in a single database LUW.

When calling lock function modules inside update function modules special care needs to be taken to provide the correct parameters. Especially the SCOPE parameter needs to be set correctly.

Erroneous update records need to be administrated and reprocessed via transaction Administrate Update Records (SM13). Regarding possible audits it is recommended to document the reprocessing of update records.

FURTHER READING:

1. SAP documentation "Update Techniques":
   http://help.sap.com/saphelp_nw70/helpdata/de/41/7af4cba79e11d1950f0000e82de14a/content.htm

4.2.2.1 Asynchronous update

Asynchronous updates are started immediately after the ending of an LUW and are executed asynchronously. Update function modules for asynchronous updates are marked with the property "Start immediately" in their attributes.

4.2.2.2 Asynchronous update in steps

Asynchronous updates in steps are started immediately after the ending of an LUW and are executed asynchronously in an own process with low priority. Therefore, an asynchronous update in steps is appropriate for the update of large data volumes if the update is not time critical (e.g. data migration programs or log data). Update function modules for an asynchronous update in steps are marked with the property "Start Delayed" in their attributes.

4.2.2.3 Synchronous update

A synchronous update is necessary if the changed data are needed immediately. This is the case for example if a subsequent LUW depends upon the result(s) of a previous LUW. The properties of the update function module need to be set analogous to the properties described in section 4.2.2.1. In order to execute an update synchronously the LUW needs to be ended with "COMMIT WORK AND WAIT".

4 ROBUSTNESS

4.3 LOGGING

Errors, exceptions and log messages in general should be stored in the business application log. This enables a central checking of all messages via transaction "Application Log: Display Logs" (SLG1). Furthermore, the transaction "Application Log: Object Maintenance" (SLG0) enables the definition of custom log objects.

The main advantages of the usage of the business application log are

1. Central Repository: the bunessiness application log provides a central repository. This central repository simplifies the administration of different applications.

2. Reusability: the business application log and the related function modules and classes provide comprehensive functionalities for logging without the need for custom development. Examples of the functionalities are:

    a. the integration of custom fields into a log object
    b. hierarchical display of messages and their aggregation into problem clusters
    c. functions to interactively read additional message information upon display of a message
    d. persistent storage of messages in the log
    e. the ability to integrate the protocol display into custom development objects (via sub-screens, controls or pop ups)

BEST PRACTICE: logging should be performed via the function modules in the function group SBAL. The example programs "SBAL_DEMO*..." provide a nice overview of the available functionality.

Please note the SAP documentation regarding the usage of the function modules in different releases.

FURTHER READING:
1. SAP Application Log – User Guidelines
http://help.sap.com/saphelp_nw70ehp2/helpdata/de/3a/c8263712c79958e10000009b38f936/frameset.htm

2. Examples for using the business application log may be found in:

Thorsten Franz, Tobias Trapp, "ABAP Objects: Application Development from Scratch", SAP Press, ISBN-10: 1592292119

4.4 PRACTICAL EXAMPLES

In the following section we describe common problems regarding robustness that are frequently criticized during software audits.

4.4.1 Incomplete CASE Statements

When using CASE statements WHEN clauses should not be left empty nor omitted completely and a WHEN OTHERS clause should always be used to handle unexpected cases.

When using IF statements the ELSE and ELSEIF clauses should not be left empty if they are provided.

4.4.2 Important SY-SUBRC Checks

The system variable SY-SUBRC should always be checked if an ABAP statement sets this variable. These checks are important in order to preserve a consistent system state even in case of an error. Especially when accessing files or the database, checking the system variable SY-SUBRC is necessary.

The following list contains ABAP statements that have not been discussed so far in this chapter. When using any of these statements checking the system variable SY-SUBRC is mandatory in order to create robust programs.

- OPEN DATASET
- READ DATASET
- DELETE DATASET
- SELECT SINGLE
- DELETE dbtab
- MODIFY dbtab
- INSERT dbtab
- UPDATE dbtab

4.5 FURTHER READINGS:

1. A list of ABAP statements that set the system variable SY-SUBRC:
   http://wiki.sdn.sap.com/wiki/display/ABAP/Sy-Subrc

5 ABAP-SECURITY AND COMPLIANCE

This chapter describes how programming errors can have a negative effect on the security of a company. Since this topic is rather complex, we can cover only a few selected central topics within the scope of this document. For further details please refer to the references at the end of this chapter.
First we need to explain the terms security and compliance in context.

We talk about a security defect when a programming error causes an unwanted side effect, through which an unauthorized user can perform malicious actions.

We talk about a compliance violation when a user can bypass an audit-relevant security mechanism of the SAP standard due to a programming error.

An important difference is that security defects in code give rise to attacks by hackers (e.g. digital industrial espionage), while compliance violations in code potentially violate legal requirements (e.g. Sarbanes-Oxley [SOX]) and/or cause red lights in financial reporting/audits.

5.1 AUDIT-RELEVANT SECURITY MECHANISMS IN THE SAP STANDARD

Before we discuss vulnerabilities in more detail, it is important to take a closer look at some central protection mechanisms in the SAP standard. These mechanisms represent core requirements for the secure operation of SAP systems.

5.1.1 Authorization Checks (A)

Roles and authorizations are a central security topic in the SAP environment. It is therefore important to understand that ABAP uses a so-called explicit authorization model. This means that authorization checks are only executed if they are explicitly programmed in the ABAP code. The best authorization concept is rather useless if custom code does not check (correctly) the necessary authorizations.

Security issues arise in the following cases:
- the developer forgets to perform an authorization check in the code.
- The developer uses the wrong authorization object.
- The developer uses proprietary authorization logic.
- The developer does not handle the return value of the authorization check correctly

5.1.2 Client Separation (B)

The SAP standard automatically separates all database access into clients. An ABAP program may only access data from the client on which the current user is working

Security issues arise in the following cases:
- the developer (intentionally) bypasses client separation by means of a technical option in Open SQL (CLIENT SPECIFIED).
- The developer uses native SQL, which does not perform any implicit client separation in general.

5.1.3 Auditability/Non-Repudiation (C)

Most business transactions must be traceable, especially in financial accounting. Custom ABAP coding must not give users the possibility to obscure audit and/or relevant actions.

Security issues arise in the following cases:
- the developer forgets to write change documents for changes to important tables.
- The developer (unintentionally) allows a so-called identity theft.
- The developer directly changes table content, i.e. without usage of standard functions which would ensure change logs.

5.1.4 Three-Tier System Landscape (D)

The SAP standard advises separate SAP systems for Development, Quality Assurance and Production. This separation primarily protects productive data from insufficiently tested programs and restricts the extensive rights of developers to the development system.

Security issues arise in the following cases:
- the code cannot be tested completely on the QA system.
- The developer uses commands that enable users to write code on the productive system.

5.1.5 Controlled Execution of Operating System Commands (E)

ABAP is technically able to execute operating system commands. Because this is potentially very dangerous, the standard provides transactions (SM49/SM69), by which you can maintain a list of allowed commands and additionally restrict these allowed commands with special authorizations.

Security issues arise in the following cases:
- the developer uses an alternative way to execute operating system commands. I.e. s/he bypasses the list of allowed commands and the corresponding authorizations.

5.1.6 Controlled Execution of SQL Commands (F)

SAP's Open SQL standard allows database access from ABAP only with a very limited set of commands, that also need to be declared statically in the code. In this way the database can be protected from dangerous SQL commands.

Security issues arise in the following cases:
- the developer uses native SQL commands to communicate with the database. In this way arbitrary commands can be executed that damage the database.
- The developer uses dynamic options of Open SQL commands and thereby allows Injection attacks.

5 ABAP-SECURITY AND COMPLIANCE

5.2 SECURITY DEFECTS

In many programming languages, application security has been an important topic for years. However, in ABAP this topic frequently is neglected/misjudged. A fundamental reason for this is the wrong belief, that SAP systems can be protected by roles and authorizations. This may be true for the SAP standard, but it is definitely wrong for custom code. As already mentioned, e.g. authorizations are only enforced if there is an explicit check in the ABAP coding.

The following types of vulnerabilities are very common in ABAP:

- missing/wrong authorization checks

  The ABAP code executes required specific authorization operation(s), but fails to check the user's authorization correctly.

- Injection issues

  The ABAP code uses dynamic commands that contain user input. If there are no input validation/output encoding measures, control characters in the input can change the semantic of the dynamic command. In this way a user can manipulate the dynamic command maliciously.

- Bypasses to security mechanisms of the SAP standard

  ABAP code must not intentionally bypass a security mechanism. Some examples are proprietary authorization checks (based on SY-UNAME), cross-client database access, and execution of operating system commands via kernel functions.

  The following vulnerabilities are very commonly observed in ABAP. Column "Std" indicates which security mechanism of the SAP standard is impaired.

ID	Vulnerability	Description	Std
APP-01	ABAP Command Injection	Execution of arbitrary ABAP Code	D
APP-02	OS Command Injection	Execution of arbitrary Operating System-Commands	E
APP-03	Native SQL Injection	Execution of arbitrary native SQL-Commands	F
APP-04	Improper Authorization (Missing, Broken, Proprietary, Generic)	Missing or erroneous Authorization check	A
APP-05	Directory Traversal	Unauthorized read/write access to files (SAP Server)	E
APP-06	Direct Database Modifications	Unauthorized write access to tables	C

ID	Schwachstelle	Beschreibung	Std
APP-07	Cross-Client Database Access	Access to business data stored in another client	B
APP-08	Open SQL Injection	Malicious manipulation of database commands	F
APP-09	Generic Module Execution	Unauthorized execution (Reports, Function Modules etc.)	A
APP-10	Cross-Site Scripting	Manipulation of a browser User , CInterface authorization/ identity theft	C
APP-11	Obscure ABAP Code	Impaired auditability caused by obfuscated code	D

Figure 1: BIZEC APP/11 – The most critical and most common security issues in ABAP-Code
SOURCE: http://www.bizec.org/wiki/BIZEC_APP11

This chapter cannot begin to provide all the necessary details for writing secure ABAP code. The only existing book on ABAP security (See entry #1 in section Further Reading) was written in German and as of this writing is not yet translated into English.

The following list contains an overview of SAP notes that describe countermeasures for some of the above vulnerabilities.

SAP Note	Vulnerability
1520356	SQL Injection
887168, 944279, 822881	Cross-Site Scripting
1497003	Directory Traversal

Figure 2: SAP OSS Notes that describe countermeasures

Of course it is recommended to analyze and install SAP security notes timely. However, these only mitigate security defects in SAP standard code. You need to check your custom code for vulnerabilities regularly.

5.3 COMPLIANCE-PROBLEMS CAUSED BY ABAP

In the past, application security has been rarely associated with compliance, but it is absolutely relevant for (financial) audits (See entry #2 in section Further Reading).

Most companies use an Internal Control System (ICS), in order to address compliance risks. This methodology is described in internationally accepted reference models like COSO (Committee of Sponsoring Organizations of the Treadway Commission) and COBIT (Control Objectives for Information and Related Technologies). In a typical ICS structure, IT General Controls (ITGC) are prerequisites for fulfilling any ICS goals in an IT-driven environment.

5 ABAP-SECURITY AND COMPLIANCE

An elementary component of IT General Controls is Change Management, which as a consequence often requires custom code. Any vulnerability in custom code represents a violation of IT General Controls and thereby compromises the very nature of an Internal Control System.

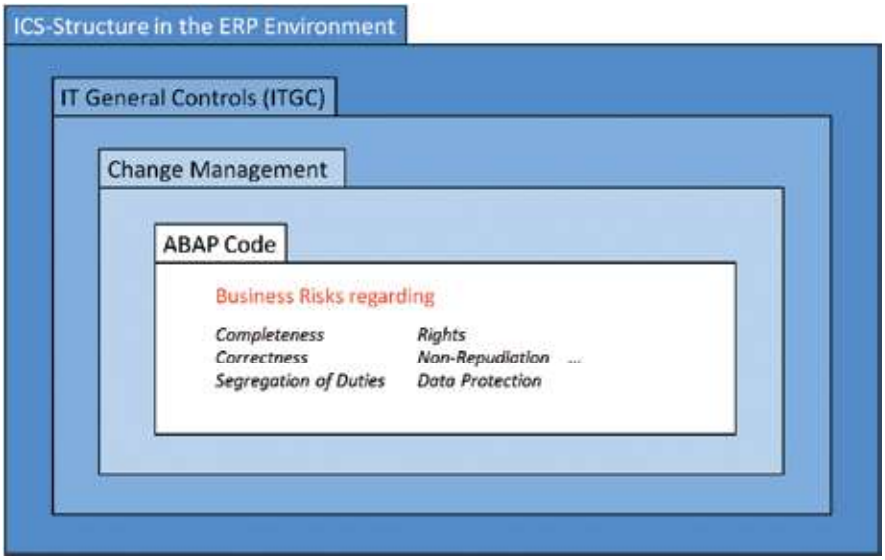


Figure 3: ICS-Risks due to insecure ABAP-Code

With particular reference to security defects in ABAP code, they can not only effect compliance standards, but potentially violate legal requirements, too.

All vulnerabilities shown in figure 1 are relevant for compliance audits, too.

5.4 TEST TOOLS

Static Code Analysis (SCA) tools are particularly suited for ABAP security tests. There are several commercial solutions that extend the capabilities Code Inspectors in fundamental areas:

- analysis of SAP standard coding, especially API calls
- very fast scan speed for Continuous Monitoring
- global data and control flow analysis, which is essential for most security tests
- comprehensive descriptions of the problem as well as proposals for solutions

- sufficient test coverage (OWASP Top 10 and SANS 25 are not sufficient since they are mostly Web-related and hardly applicable to ABAP) (OWASP is the Open Web Application Security Project).
- four-eyes principle for handling exceptions

Of course, such a tool must be integrated fully into SE80, TMS and ChaRM, so developers are able (and willing) to work with it.

5.5 FURTHER READINGS:

1. Andreas Wiegenstein, Markus Schumacher, Sebastian Schinzel, Frederik Weidemann, Sichere ABAP Programmierung (Secure ABAP Programing), SAP Press 2009
2. Maxim Chuprunov , Auditing and GRC Automation in SAP, Springer 2013
3. BIZEC – The Business Application Initiative (http://bizec.org)

6 DOCUMENTATION

In many cases software documentation is as important as the development itself. If no documentation exists or if it is not comprehensive enough, this can lead to increased efforts once additional development is required or if there is a change in developers. This chapter describes different options of how to document development objects in the SAP system.

6.1 DOCUMENTATION INDEPENDENT OF DEVELOPMENT OBJECTS

In addition to the description of the many development objects performing specalised functionalities in the ABAP system, there is also the need to document the relationships within and between the modules. Questions like the following need to be answered:

- Which dependencies exist between the modules?
- How can the processes be mapped to reports?
- What processes are executed when during a day/month/year and what development objects are impacted?

In our opinion there is no suitable repository within the SAP development system, which also accommodates graphics easily, to help answer these questions. For the documentation of overarching relationships we therefore recommend the utilisation of non-SAP tools.
Examples are:

- Internal (Product) Wikis
- Documents in well maintained public repositories (portal storage, Sharepoint, fileshare ...)

Experience shows that the real challenge in this area is primarily a question of discipline. No tool can solve this challenge. This is up to the development team and its management.

It also needs to be stressed that obsolete documentation can be very confusing. Because of this, the documents should contain some information about their status and version. This will help with evaluating how current the documentation is.

Within an SAP system landscape the SAP Solution Manager provides options for project documentation. The links listed below lead to additional information:

ADDITIONAL REFERENCES:

1. Help.sap.com Dokumentation SAP Solution Manager 7.1:
   http://help.sap.com/saphelp_sm71_sp05/helpdata/en/3d/d05893e6ba4dfab7c0d66de8d52420/frameset.htm
2. SCN Blog: "Business process documentation with SAP Solution Manager 7.1"
   http://scn.sap.com/blogs/ben.schneider/2011/11/04/business-process-documentation-with-sap-solution-manager-71

6.2 DOCUMENTATION OF DEVELOPMENT OBJECTS

Some development objects like methods, function modules and reports can contain documentation directly within the source code. Others do not have this option and have to be documented differently. Examples are:

- Interfaces
- DDIC-objects
- Transactions

BEST PRACTICE: we recommend that all development objects – independent of the source code – make use of the documentation options provided by the ABAP workbench and to document the tasks and importance of the objects within the SAP-system. The external to the source code documentation should describe the current state of the processing whereas the documentation in the source code is focused on the applied changes and their sequence.

As the documentation is connected with the transport system, it will be available in all instances of the system landscape. In addition, the documentation can be accessed by all users and in some cases (reports), the ABAP system automatically integrates it into the user interface. This documentation can also be translated, which is yet another advantage for using it.

6.3 DOCUMENTATION IN THE SOURCE CODE

6.3.1 Documentation and comments of statements and processing blocks

Generally speaking, processing blocks should briefly be documented in the source code to make it possible for people not familiar with the program to nonetheless quickly gather the main points of the code. The goal is to describe, if possible in one comment line the purpose of the subsequent processing block. The focus is on why something is coded the way it is instead of how it is done. The following rule applies: as little commentary as possible, as much commentary as needed. Therefore, avoid redundant information (like the name of the documented module or the name of who applied the changes).

BEST PRACTICE: English should be used for the comments. Today, development work is most often done in international teams and even if you currently do all your development in another language, there is always the likelihood that your project will become more global in nature over time. The later effort will then run into coordination issues, or have to re-engineer translations, and this will be more difficult than the possibly somewhat larger initial effort of writing the documentation in English.

It has also been shown, that the readability of the code and comments is improved if the comments are in English. The reason for this is that the ABAP statements are also English-like and have a structure similar to sentences. Anybody reading the English documentation in the code then does not have to keep switching back and forth between different languages.

6 DOCUMENTATION

6.3.2 Change documentation

Once a program has been moved into production, all the subsequent changes need to be documented in the program.

6.3.3 Program header

Changes in programs are documented in the program header with the initials of the developer (e.g. built from the first and last names), the date the change occurred and the related change document. Ideally, the initials are also explained in the header. It also helps, however, if the explanation is added where the changes are implemented. If this is done, the reason for the code change can be understood easily.

Example:

*/ Change Log

*/ VN/Date
ChangeDoc Description

*/ MZ/2012-08-06 CD4712 Add MMSTA in Output, Max Mustermann

*/ MZ/2012-02-01 CD4711 Import Material number, Max Mustermann

*/ MM/2009-01-01 CD0815 Added Field ABC in method signature and source code in order to support quick search, Max Mustermann

The description of the change should answer the question "Who changed what, why and how?".

If change-management or bug-tracking-systems are used for the coordination of further development work, the comment should contain references to those incidents/issues. This will later allow one to identify straight from the code which enhancements or bug fixes caused a change.

BEST PRACTICE: the comment is not meant solely for the developer who implemented the change, but for other developers who follow or perform quality reviews. Please keep this in mind whenever you write or review comments.

Lines of code – optional

Changed lines of code can be documented with a combination of the initials and date of change.

Obsolete change-documentation, should be deleted to maintain the readability of the code. The overarching goal should always be that the source code is easy to read.

Simple change

```
*WRITE: lw_mara-matnr.  "MZ/2012-08-06 CD4712 CD4711 Add MMSTA in Output
WRITE: lw_mara-matnr, lw_mara-mssta.
```

Change of a statement block – Deletion of several statements

```
"--> MZ/2012-02-01 CD4711 Import Material number
*CONCATENATE   wa_import-aufnr wa_import-vornr
*              INTO str SEPARATED BY , ..
"--> MZ-2012-02-01. DELETE
```

Change of a statement block – Insertion of several statements

```
"--> MZ/2012-02-01 CD4711 Import Material number

CONCATENATE   wa_import-aufnr wa_import-vornr wa_import-matnr
              INTO str SEPARATED BY , ..
"--> MZ/2012-02-01. INSERT
```

Using the asterisk or double quotes for comments

Asterisk-comments should only be used in the program header or to comment out old code.

SAP recommends to us inline comments for everything else. These should be placed above the code to which they refer and they should follow the code's indentation.

ADDITIONAL RESOURCES:

Keller, Thümmel, ABAP-Programmierrichtlinien, SAP Press 2009

7 FEASIBILITY AND ENFORCEABILITY

This chapter describes how the best practices from this guideline could be realized in practice. We differentiate between Feasibility and enforceability.

In the section "Feasibility", we explain what companies have to consider when they want to introduce programming guidelines. We describe how a process might appear, how to launch it and last but not least, how to keep it up to date. In the section "Enforceability", we demonstrate how the company could check the specifications of the process. For these organisational aspects test methods and tools have to be considered. We also examine the limits of enforceability.

In conclusion, we describe tips fom practical work experiences which have been collected by the authors in several projects in SAP development environments.

7.1 FEASIBILITY

Whoever wants to implement programming guideline successfully in a company must involve the management in the process. The reason for this is that the improvement of code quality requires as a first requirement an investment in processes and tools as well as in training of people involved. In particular, the management has to be assured that the company will save costs on this process in the long run.

7.1.1 Motivation for a Process

In the following section you will find guidelines, which quality aspects have to be addressed during the implementation of a quality assurance process and which advantages this brings to the company.

Security
Advantage: the company avoids that users access or change critical data without authorisation.
Risks in case of quality deficit: sabotage, industrial spying, unwanted press publication produced by data leaks and downtime of productive systems.

Compliance
Advantage: the company can prove at any time that the developed software will be adequate regarding relevant compliance standards and legal requirements.
Risks in case of quality deficit: financial audit failures, violation of compliance or legal requirements (e.g. data security).

Performance
Advantage: the company ensures that the hardware could be used in an optimal way and therefore protects the previous investment in hardware. In addition, the employees' satisfaction increases, because the use of the application will be more productive.
Risks in case of quality deficit: user acceptance decreases or there will be costs for faster hardware to try and compensate for software defecits.

Robustness
Advantage: company ensures the continous operation of the business application and avoids unproductivity in case of system failure.
Risks in case of quality deficit: user acceptance decreases and the operation costs rise because of user unproductivity, error analysis and technical maintenance.

Maintainability
Advantage: company ensures that the application will be maintained cost-effective, because the program structure is easy to understand and well documented.
Risks in case of quality deficit: high maintennace costs and generally a higher error rate of the application.

7.1.2 Design and Maintenance of the process

For the implementation of this method in practice, a formal description of a process has been done. This includes clear procedures and responsibilities. The precise design of the process is company-specific and cannot be described here, the need however, is universal.

BEST PRACTICE: define the process to be followed and document it so that everyone can access it. Define how changes and improvements of the process should take place and how comments and criticism can be given. Document all tested rules in detail, with chapters for background/ motivation, bad examples, good examples, information on the procedure for removing, literature and the contact person(s) in the company.

Motivation
A development best practice process helps to improve the quality of software proactively and efficiently and to reduce costs in the company in the long term. The earlier an error is detected during development, the easier and more cost efficiently it can be corrected. The fewer mistakes included in an application, the more the benefits correspond to the expectations of the company. In particular, it runs without any adverse side effects that have a negative effect upon the business.

Which aspects are relevant for the process?

Internal Development
For internal development, guidelines are needed as a reference for daily work and regular training on actual risks.

External Development
For external development very clear quality specifications are necessary for the bidding process. Before approval, the requirements must also be checked.

Overall
The process requirements must be checked as early as possible with suitable tools. Frequently, an overall manual test is not possible.

7 FEASIBILITY AND ENFORCEABILITY

BEST PRACTICE: specifications without a tool-supported verification option cannot be implemented and should not be defined nor specified.

Each rule, which shall be used for subsequent QA, must be defined how it can be verified with tool support. If no tool-supported inspection is possible, it will be nearly impossible in practice to ensure compliance with this rule. Concentration on tool-supported verifiable rules spares those who are responsible for quality assurance, of a number of frustration sources.

Nevertheless, there are several aspects that elude automated testing. These aspects normally can be covered only by regular code reviews. Since well conducted code reviews require significant effort in their implementation and the preparation and review or control of corrections, they must be limited to non-tool-supported, verifiable and critical development objects. If performance requirements compliance have a high priority, code reviews should be restricted only to development objects with access to database reads or extensive calculations.

Therefore, the process must provide quality assurance, by which all developments will be tested before they are transported into production. It must be defined, how to deal with errors, too. Of course this process must be updated regularly to support new aspects.

7.2 ENFORCEABILITY

7.2.1 Manual Test

Many tests can be automated. However, there are areas that are not suitable for automatic testing, such as documentation, architecture or many functional requirements. Language is complex, therefore, the content of documents and documentation must be checked manually. Only a human reader can judge whether a text is meaningful, complete, comprehensible and correct. An automatic test (to date) can only check for the existence of the provided languages. However, it is still recommended to use an automatic test on the non-functional aspects.

For manual testing, complete test by analysis of transport lists is prefererable. It is important to consider what internal guidelines exist. Depending upon the number of objects, there must be a full or at least a random test. The test result has to be sent to the developer/person responsible for the improvement/completion of documents/documentation.

In practice, a regulary planned cyclical review of the process has proved its worth.

When and how should be tested?

The concepts upon which the tests are based must be checked regularly for requirements, relevance and compliance. The up to date status has to be ensured by new release (Enhancement Package) upgrades. Concerning specifications, it is quite usefull to consult the auditors for the company. Regarding externally developed code the tests have to take place before the acceptance.

For the acceptance of tests or the compliance part of the manual tests, it is useful to carry out the necessary tests manually during development and QA tests (four eyes priciple) by different developers.

The same applies for (security) penetration and for load tests. Since penetration tests are a critical security issue, it may be necessary to consult external partners for this purpose regularly.

7.2.2 Automatic Tests

Automatic tests quickly cover a large part of the necessary tests and examinations. Scheduled as a background job, repitition with little extra effort is possible. These regular tests carried out with the same quality enable the developers to improve their programming style.

When and how testing should be done?

The developer should receive feedback regarding the conformity of the developments with the guidelines as soon as possible. For this purpose daily scheduled tests in development system are useful and the developers should be provided with the results. It is important to use the same tests and metrics for each developer during the test of the central QA instance and the transport release. If different tools or different settings are used, the acceptance by the development team decreases significantly.

As a central protection instance the tests must be implemented in TMS and at latest at transport release (or even better at task release) time. This ensures that neither unchecked developments nor developments not meeting the guidelines are transported into other systems or into the production system.

As a "last security net" a regular test run (full scans) in the production system should take place. This should be planned during a low-load time via a background job. The result is provided to the QA responsible person who arranges the further high priority steps (and when necessary, corrections).

For all automated tests it should be defined in advance how to deal with old code. It makes sense to create a timetable when and how the new rules are applied to old code.

7 FEASIBILITY AND ENFORCEABILITY

7.2.3 Tools

Some of the tools for automated tests are listed below.

SAP Code Inspector

SAP Code Inspector is delivered by SAP as a standard and is therefore highly integrated into the development environment. SAP provides an extension concept in which it is possible to implement custom checks. Therefore, programming is necessary. The tests delivered by SAP are in some parts extensively parameterized. Inspections of large sets of objects can take place both online and in batch mode. The result sets of previous inspections can be used as an object set to limit the corrections to those of the previously identified faulty objects. Furthermore, results can also automatically distributed by email to the responsible developer.

Werkzeuge von Drittherstellern

In addition to the SAP Code Inspector, there are very good commercial tools on the market, that carry out tests at the code level. A description of these tools is omitted for neutrality reasons.

7.3 PRACTICAL EXPERIENCES AND TIPS

7.3.1 Source Code Quality Assurance

To ensure a successful implementation, it is important that quality assurance is carried out step by step and carefully. It is advisable to take a two part approach. First of all new code should be created "error-free" and this has to be checked. Only when this process has stabilized, should existing code be added gradually to the tests, otherwise there is too much work for the developer and the motivation drops rapidly.

If new development is not started from scratch, when automatic code reviews are introduced, it is important to clarify how to deal with the exsisting code. Even with new developments changes to existing objects cannot be avoided, which leads to problems, if transport checks are implemented. In such cases a clarification of the responsibility for development objects is helpful, which can be documented in the appropriate field in the package definitions. These responsible persons must decide whether errors in existing code must be corrected immediately or if exceptions are possible.

In many cases it is sufficient to work with standard on-board tools, such as Code Inspector, which could be extended with custom checks and thus could be adapted to one's own needs. As of NetWeaver 7.02 some new tests are included in delivery, such as search for certain critical instructions, usage of the ADBC interface, client specific shared object methods, robust programming and ABAP WebDynpro metric.

7.3.2 Time and Budget QA

To keep the time and effort for the QA activities to a minimum, developers must have the opportunity to examine the code independently during its development for errors.

If s/he does not,s/ he must automatically receive feedbacks on errors or opportunities for improvements. Daily inspections with an appropriate tool and distribution of results ensures that errors are detected early and the developers can still remember their activities from the day before, which significantly simplifies troubleshooting. Hereby, it is ensured that even developers who do not want to make manual work or are under pressure receive the chance to fix their coding errors. N.B., the later QA takes place, the higher the costs for bug fixing. This additional effort arises for example by additional transports if the original transport has been released.

Therefore, it is important to consider the quality of project planning and estimation not just at the end, but parallel to the project, and subsequently in the complete software lifecycle.

Do not underestimate training effort, which is required to convince the developers of the necessity of the process.

If third-party developers are employed the programming guidelines and naming conventions have to be part of the contract.

7.3.3 Problems

With the introduction of a code QA a number of issues occur, which will be discussed briefly here. A conflict occurs around the question of who is responsible for QA, creator or changer. For new sources, this is not a problem, but for existing code, this question always arises:

"Why should I check Code in which I only changed one line?"
vs.
"Why should I check code, which I haven't touched for years?"

Both positions are of course understandable, so there must be a clear decision regarding handling of copied code, which is not based on other/different/no conventions .

BEST PRACTICE: to answer questions of developers, when problems occur, it is important to create a central authority. In addition, there must be a process to release even faulty transports in case of an emergency. If this possibility does not exist, the acceptance for the step decreases. One possibility is to install an approval process for the release or transport of faulty code. Most third party tools on the market offer this option by default.

7 FEASIBILITY AND ENFORCEABILITY

7.3.4 Decision Making for Modification

It is important to set the threshold for modifications as high as possible. This is especially important if you decide to implement SAP enhancement packages as soon as possible (SPAU problem). The starting point for this is the modification key. The number of developers, who have permission to create a key modification, must be as low as possible. Thus, it is possible to control modifications by an appropriate process with change requests.

The question, when a modification is reasonable has to be answered by each individual company and has to be implemented consistently. An answer to this question fitting all cases does not exist. In each case decisions should be based upon equal, defined in advance and communicated criteria.

7.3.5 Practical Field Report: Comgroup GmbH

The following example of Comgroup GmbH, IT service provider of the Würth Group, shows how programming guidelines and naming conventions in software development could be implemented and enforced.

Comgroup GmbH started with the code QA in a global development of a multistage system development landscape. For automated support Code Inspector was used. Since code QA introduction coincided with introduction of a new namespace, at the beginning of the project only new namespace objects were checked, existing code was not considered. This facilitated the selection in Code Inspector, because Code Inspector did not need to consider change nor create date in selections. In addition, Code Inspector performance and security checks were not initially activated to keep the effort within manageable limits.

In this way the developers have the option to check their code during development activity independently. In addition, a nightly run was scheduled to analyze the complete relevant code and send emails in case of errors. The problems experienced here were that there were many users in the system, with no associated email address, which meant that at the beginning emails did not reach their recipient.

Emails to developers without an email address will be shipped to a central address and thus the incomplete data can be identified with a little effort. Therefore, it is no longer allowed to create development users in the system without an email address.

The emails were not sent by an anonymous batch user but by the email address of the responsible person for QA in order to give the developer the opportunity to ask questions via a reply. In the beginning, this resulted in a lot of effort, but the number of questions decreased during the project. This was achieved by ongoing training, during the project, so that the developers could efficiently correct the errors or avoid them during development.

The development language in development landscape is English. This was checked by a further job and the errors were reported to the developer. SAP offers no way by default to set the correct language for the system. Therefore, there was only the possibility to implement a check by modification or to live with the fact that objects were created in the wrong language and had to be changed afterwards.

System naming conventions of objects were checked during creation so that they could only be named according to the conventions (also done via modification).

To perform custom checks at transport release (e.g. own naming conventions/at a minimum German and English translations) an implementation of the Business Add In CTS_REQUEST_CHECK was created and CHECK_BEFORE_RELEASE method were used.

After the process had stabilized in the global development system, it was rolled out to the subsequent development systems and namespaces. Existing code has not to date been checked. In addition, it is planned to use an external tool which simplifies quality assurance.

8 INFRASTRUCTURE AND LIFECYCLE MANAGEMENT

Apart from methodical recommendations and tools used for software development in a SAP System the infrastructure and the lifecycle management of a software component are important factors. This chapter focuses on these topics.

8.1 INFRASTRUCTURE

A SAP system landscape usually consists of several separate systems. In the following sections the different systems of a sap system landscape are described in detail.

8.1.1 Sandbox System

A sandbox system is only used for testing and exploration of ideas. In a sandbox system no restrictions regarding authorisations, customizing and development are enforced. No transport requests are allowed from the sandbox system into any other systems of the system landscape. The sandbox system is used to test ideas prior to implementing them in the development system. In a sandbox system complex developments do not need to be removed if their development is discarded.

8.1.2 Development System

The development system is used to perform development and customizing.

A combined development and test system (e.g. development client 100 and test client 200) might be a sensible option for a simplified system landscape. In such a system no transport requests are required to test new developments.

In a combined development and test system no customizing must be performed in the test client. The customizing of the test client is only updated via a client copy (transaction SCC1).

Developers have extensive authorizations in the development system. However, certain restrictions need to be defined individually, e.g. to restrict the access to sensitive data like HR data.

Generally, all changes that can be transported are only performed in the development system. This is also true for authorizations and authorization roles. These are also created in the development system and transported into the subsequent systems.

8.1.3 Quality Assurance System

The QA system is used to perform tests after any customizing changes or for new developments. The QA system can be created by a system copy of the production system. This approach ensures the equivalence of the test and production data.

Absolutely no customizing, nor development is performed in the quality assurance system (QA system). Changes to the customizing and the development objects are only performed by importing transport request. Imports into the production system are only performed after the import into the QA system was performed. This ensures a system environment that is equivalent to the production system.
Developers have extensive authorizations in the development system. However, certain restrictions

need to be defined individually, e.g. to restrict the access to sensitive data like HR data. However, it is advisable to use test users with authorizations similar to the production system for tests in the QA system.

8.1.4 Production System

Absolutely no customizing nor development is performed in the production system. Changes to the customizing and the development objects are only performed by importing transport requests.

Developers only have very restricted (if any) authorizations in the production system. The usage of emergency users needs to be defined, both technically and organizationally. Data changes (e.g. via &SAP_EDIT) are only performed in an emergency situation and need to be documented with a date and a justification. It is advisable to use a tool to standardize the documentation.

8.1.5 Transports

The technical release of transport requests should be performed by responsible internal employees in general and not by external consultants. Prior to the technical release a formal approval (refer to change management) is necessary. Transport requests are imported centrally by SAP basis administrators. The only exception to this rule is the client copy within the integrated development and test system.

The transport route is defined as follows:
Development system -> QA system -> production system.

The sandbox system is not part of the transport routes. Imports into the sandbox system are only performed for individual requests. For these imports the transport route is always development system -> sandbox system, never the other way around.

Imports of transport requests into the QA system need to be organized depending upon the system landscape. If possible, the imports into the QA system should be automated in order to avoid unnecessary work for the basis administrator(s).

Imports into the production system need to be approved individually. For this approval the usage of an appropriate tool (e.g. SAP Solution Manager) is recommended in order to standardize and formalize the process.

BEST PRACTICE: the following check list prior to the release of a workbench transport request:

1. Have tests been performed? All functionality was tested and approved by the user/customer

2. Was the code checked? The code inspector checks have been performed for all program code in the transport request. No error nor warning type messages are allowed in the result list. Information type messages are acceptable.

3. Third party tools? If additional third party check tools exist, e.g. security checks or performance checks, make sure that these have been executed.

4. Manual tasks necessary? Is a complete check list for all necessary manual tasks available?

5. Internationalization? Are all necessary translations present in the transport request?

6. Transport dependencies? Dependencies between different transport requests need to be checked

7. Naming conventions? Have the defined naming conventions been applied for all objects in the transport request?

8. Documentation?
   a. System internal: check all objects regarding the creation/update of the SAP documentation, e.g.:
      - report documentation (description and comments in the ABAP code)
      - function module documentation (parameter, exceptions, etc.)
      - data elements, structures, tables (descriptions, short and long texts, etc.)
   b. Outside of the system: is the required documentation available, complete and approved?

8.1.6 Removal of obsolete developments

If development objects have been created in the development system only or have only been transported into the QA system but never into the production system, these obsolete development objects should be removed from the development system. If the TMS Quality Assurance workflow is used to approve transports the following steps need to be performed:

- Rejecting a transport request only inhibits its import into the production system. However, the rejected development objects/customizing are still available in the development and QA system. Consequently, the rejection of a transport request results in an inconsistency between the development and test system on one hand and the production system on the other hand.

- To correct this inconsistency the rejected development objects/customizing need to be removed in the development system, added to a transport request, and imported into the QA system.

- The Transport request containing the removal of the development objects/customizing should also be rejected in the QA system. Thereafter, the consistency of the system landscape is restored.

- The developers responsible for the rejected transport request need to be informed about the rejection. The person responsible for the approval of transport request in the QA system should also be responsible for informing the developers.

8.1.7 Safeguarding the consistency of developments

In parallel projects the danger of overlapping developments exists. E.g., development objects not present in the target system might be already used in a different development object. This situation results in errors during the import of transport requests. Consequently, custom development objects need to be checked prior to their usage. In order to safeguard the consistency of developments the number of transport requests should be restricted to one transport request for workbench, customizing and authorization roles respectively.

Transports into the QA system should only be performed using transport of copies. The final release of a transport request should only be performed at the end of a project. All developers involved in a project only use tasks inside the single transport request of the project. Individual transport requests of developers should not be used.

In general, the import of a transport request is only performed after a formal approval by the process owner or the responsible tester. The exact process as well as the involved departments needs to be defined individually for each company.

8.2 CHANGE MANAGEMENT

In order to keep a SAP system landscape maintainable and controllable, a formal change management process is required. This change management process is independent of the kind of the change. It should be applied to customer specific developments as well as to changes in the SAP standard.

The basis for the introduction of a change and release management process is the Information Technology Infrastructure Library (ITIL). This is a reference guide that contains a comprehensive list of generally accepted best practices.

This section describes a specific user case of a change management process in the area of software development. This use case can be adapted to the specific need of different companies and industry sectors.

In general, the following aspects need to be taken into account when introducing a change management process

- functional requirements
- motivation
- evaluation (effort estimation)
- acceptance
- approval of the change for the production system

8 INFRASTRUCTURE AND LIFECYCLE MANAGEMENT

Example of a change control document:

Change Request / Change Control

CC Number: _____ Date: __ / __ / ____

CC Title : _____

Request

Requester: _____ Cost Center: _____

Department: _____ Priority: ( ) low ( ) medium ( ) high

Preferred Date: __ / __ / ___ Type of CC: ( ) Change ( ) Authorization

Short Description: _____

_____

_____

(please add an additional, detailed description as attachment)

Process Owner: _____ Date/Signature: _____

Approval

Appl./Module: _____ Person in Charge: _____

Approval SAP CC _____

Approval IT Mgmnt _____

Comments: _____

_____

Release to Production Environment

Verified by Requester _____ (date/signature)

Released by SAP CC _____ (date/signature)

Released by IT Mgmnt. _____ (date/signature)

Transported by _____ (name/date/signature)

Figure 4: Change control document

The change control document shown above provides an example of the essential data required in a change management process

Change process and roles:

- The requestor of a change fills in the part "Requesting Department" of the change control document. Furthermore, s/he is responsible for obtaining the signature of the process owner and the external process owner.

- The process owner is usually the line manager of the requestor. S/he is responsible for certain parts of the data in the system or certain parts of the SAP software. For example, the purchasing manager is responsible for SAP applications and data related to the procurement process.

- The external process owner needs to be involved whenever the change effects areas of the SAP software for which the process owner is not responsible personally.

  As an example consider the case where the procurement manager requires authorizations in the assets accounting. In this case the process owner of the asset accounting needs to be involved in the change.

- A complete description of the change needs to be added to the change control document as an attachment. Incomplete change control documents are rejected.

- The requestor passes the change control document to the IT department.

- The SAP coordinator is responsible for coordinating all activities in the SAP solutions or parts of an SAP solution. The SAP coordinator assigns tasks to the developer responsible for implementing a change in a certain SAP module. The role of the SAP coordinator can either be assumed by a single person or a team, depending upon the size of the organization. The SAP coordinator adds the application/module to the change control document and assigns a responsible person for the change. The SAP coordinator also might reject a change document based upon formal errors (e.g. insufficient description of the change or a missing aprovement of the process owner). The SAP coordinator assigns a unique change number and a change title to the change control document. The change number could, e.g., be taken from a project management tool.

- The head of IT or head of SAP approves rejects or defers (with a suitable justification) a change after it has been approved by the SAP coordinator.

- In the case of an approval the responsible developer receives the change document for further processing. Developers are not allowed to perform any changes without a complete and approved change control document.

- After implementing the change, the developer requests a test by the requestor.

- Is the requirement implemented correctly, the requestor approves the change for importing into the production system. The requestor approves the correct implementation of the change with his signature. A transport request is never imported into the production system without the approval of the requestor.

- The SAP coordinator and the head of IT confirm the correct implementation of the change. A transport request is never imported into the production system without the approval of the SAP coordinator and the head of IT.

- The responsible developer releases the transport request for importing in the production system and finally forwards the change control document to the SAP coordinator and the head of IT.

8 INFRASTRUCTURE AND LIFECYCLE MANAGEMENT

The required approval process and consequently the contents of the change control document- might vary significantly across industry sectors. In the pharmaceutical industry for example, the QA department is always involved in the change control process. In addition to that, changes exceeding a certain budget usually need to be approved by additional committees. The structure of the approval process depends upon the organizational structure of the company.

Consequently, the described change control document only shows the minimum set of requirements without any industry-specific nor organizational-specific additions.

The addition of fields or references to further documents (e.g. validation documents) to the change control document needs to be implemented on an individual basis. The same is true for the extension of the approval process with additional approval steps.

FURTHER READING:

1. Mathias Friedrich, Torsten Sternberg, Change Request Management mit dem SAP Solution Manager, SAP Press, 2009
2. Information Technology Infrastructure Library (ITIL)

8.3 MAINTAINABILITY

Maintainability of software is a criterion in software development. It measures how much effort is required to change an application in the overall system landscape (source: Wikipedia).

Maintainability requires modular objects according to SAP best practices. This includes the usage of APIs that can be reused by other developers.

In system landscapes consisting of different development and production systems (different development streams) an important rule is that an identical object name (transaction code, program, class or table) provides identical functionality. Different functionality needs to be encapsulated in a different object or in customizing.

All development, changes and bug fixes need to be documented properly.

8.4 ADAPTATION OF THE SAP FUNCTIONALITY

In order to adapt the functionality of a SAP system to the individual requirements different approaches are available. Each of these approaches has certain advantages and disadvantages. The available approaches are:

- Modification
- Copying of a development object into the customer namespace
- Enhancements (user exits, customer exists, BAdIs, explicit and implicit enhancements, Business Transaction Events)

User exits, customer exits, Business Transaction Events and BAdIs can be used without any problems with respect to manageability. Therefore, it is advisable to use these technologies for the adaptation of the SAP software whenever possible.

User-Exit

User exits are sub-programs contained in the SAP namespace. They are only delivered once by SAP and can therefore be "modified" without any problems.

Customer-Exit

Customer exists are function modules that can be activated and deactivated. A custom implementation of the function module allows the adaptation to the standard functionality.

Business Transaction Event (BTE)

The FI applications' BTEs are an additional possibility to adapt SAP functionality. BTEs are similar to customer exits. In contrast to customer exists they are restricted to the FI area and provide a predefined interface. Further information on BTEs can be found in the SAP documentation.

Business Add-In (BAdI)

With BAdIs SAP tried to remedy the limitations of the aforementioned technologies.
These limitations are:

- only single usage (customer exits)
- no dynpro enhancements (BTEs)
- no menu enhancements (BTEs)
- no maintenance tools (BTEs)

Therefore, multiple implementations can exist for one BAdI and all enhancement types (program, menu and dynpro) are possible. Furthermore, BAdIs are implemented using ABAP OO.

If the required result cannot be achieved using the aforementioned approaches, further approaches to enhance the SAP standard are available. However, their usage needs to be checked on a case-by-case basis.

Enhancement Spot/Enhancement Section

Enhancement Spot:
Enable to inject ABAP code at predefined (implicit and explicit) spots in the program.

- multiple active implementations
- all active enhancement implementations are executed.

8 INFRASTRUCTURE AND LIFECYCLE MANAGEMENT

Enhancement Section:

- multiple active implementations
- only one active implementation is executed
- if multiple active implementations exist it is not defined which active implementation is executed

Note: implementations of enhancement sections might be disabled when implementing SAP enhancement packages. Furthermore, the activation of a business function might substitute the current implementation with a new active one or new implementations might be activated. In such situations it is very difficult to identify substituted or no longer executed enhancement implementations. Therefore, a change in the SAP standard might change the behaviour of the enhancement. This fact increases the necessary test effort significantly and may lead to severe problems during a SAP upgrade or during EhP implementations. Therefore, the usage of enhancement sections needs to be carefully evaluated on a case-by-case basis.

If enhancement spots or enhancement sections are used, the transaction SPAU_ENH needs to be executed during an upgrade. The reason is that those enhancements are not displayed in the general SPAU transaction.

Consequently, the decision if enhancements should be used needs to take not only the implementation effort, but the subsequent effort during upgrades into account, too.

Modifikation

In general modifications should only be performed if:

- the requirement cannot be implemented using customizing
- suitable enhancements or user exists are not available
- copying the SAP object into the customer namespace is not sensible.

For modification, exits and BAdIs a separate approval process in addition to the change process is advisable. This process consists of a request, a justification and an approval. The decision for an approval needs to be made by the person or team responsible for the system (e.g. IT management). Modifications require a modification key. These keys are administered centrally. This is to make it easy to prevent unwanted modifications in the SAP systems.

Copying into the customer namespace

Copies of SAP standard programs in the customer namespace require high maintenance efforts. Currently, no automated tools nor manual best practices for a subsequent alignment (i.e. in the context of an upgrade or when implementing a SAP note) of the original and the copy exist. Therefore, general best practices cannot be provided here. The advantages and disadvantages need to be carefully considered on an individual basis.

BEST PRACTICE for conducting a modification:

- Generally modifications of workbench objects should only be performed using the modification assistant.
- A copy of the object into the customer namespace should not be the first choice, as it results in high implementation efforts. Furthermore, improvements in the SAP standard are not automatically integrated into the copy. Therefore, further effort is required to realign the copy with the standard. Finally, the implementation of enhancement packages might result in problems related to standard includes.
- The decision between modification, copy in the customer namespace and enhancements not only depends upon the implementation efforts but also upon the maintenance efforts.
- Each of the different options has different advantages and disadvantages. Therefore, the best approach needs to be evaluates on a case-by-case basis.
- It is advisable to create a central, formal, technical documentation of all modifications. Suitable templates should be provided for enhancements and copies. Their usage should be mandatory.

FURTHER INFORMATION:
SAP University Courses BC425 and BC427

8.5 TESTABILITY OF APPLICATIONS

In order to ensure the testability of applications, the test requirements need to be determined early in the development process. To enable the testability of existing code usually requires large effort. Furthermore, enabling testability does not result in new functionality and is therefore often seen to be of a low priority.

In order to efficiently test an application, it is necessary to automate the tests. In order to automate testing, the execution of tests needs to be repeatable. Program code should be written in such a way, that it can automatically be analysed by static code analysis tools. In order to enable static code analysis dynamic expressions should be omitted. The reason is that the semantics of a dynamic expression are only known during execution. Therefore, it usually cannot be analysed by static code analyses tools.

BEST PRACTICE: automatic tests tools (ABAP Unit, eCATT) should be an integrated part of the development process.

BEST PRACTICE: the test workbench provides the tools for test-driven development via unit test. Unit tests are implemented as local classes. The local classes are marked via the expression "FOR TESTING" as test class. The code implemented in the local test class is only executed via the menu item "unit test" in the ABAP workbench. So called risk levels define the criticality of a unit test. The system settings might prevent the execution of a unit test of a certain risk level.

8 INFRASTRUCTURE AND LIFECYCLE MANAGEMENT

In the context of ABAP programs the tight integration of database accesses is an impediment for repeatable automated tests. In these cases database entries need to be created or adjusted prior to the test execution. After a test, changes to the database need to be removed in order to not hinder subsequent tests.

BEST PRACTICE: separate database access as well as calls to remote systems, whose runtime behaviour cannot be controlled, forms the core of the application. As soon as the application core does not access the database nor remote systems directly but via interfaces, these interfaces enable the simulation of the database or remote system access for testing.

If, for example, all database accesses (SELECT, INSERT, UPDATE, DELETE) should be encapsulated in a database access layer. This abstraction of the database access allows executing unit test, using defined and consistent test data.

Further Information:
www.testbarkeit.de
http://de.wikipedia.org/wiki/Testbarkeit
http://www.testbarkeit.de/Publikationen/TAE05_Artikel_jungmayr.pdf

9 THE AUTHORS

The following authors were heavily involved in creating the current version of these guidelines:

Peter Lintner, Senior Consultant, Allgemeines Rechenzentrum GmbH

As a certified project manager (IPMA Level C), Mr. Lintner has been working in SAP ABAP development since 1998. His areas of focus lie in the application and workflow development as well as in the change and request management.

Steffen Pietsch, Vice President, IBSolution GmbH

Mr. Pietsch has been working in a development-oriented environment since 2003 where he gained experience as a developer and additionally experience in multiple managing positions. As a spokesman for the DSAG working group SAP NetWeaver Development since 2009, he has represented the interests of customers and partners in cooperation with SAP.

Markus Theilen, IT Coordinator, EWE AG

Since 2001, Mr. Theilen has been working as a developer, software and enterprise architect. In these functions, he gained extensive experiences in complex SAP ERP implementations. Since 2012, he, has been IT coordinator in the EWE Group, and he has been managing the development activities of specific applications.v In addition, he works as a deputy spokesman of the DSAG working group "SAP NetWeaver Development" in DSAG e.V.

Jürgen Wachter, Process Coordinator Development, Comgroup GmbH

Since 2002, Mr. Wachter has been working in the area of SAP development. His professional focus lies on core development/enhancements

Michael Werner, SAP Application consultant (Inhouse), LTS AG Andernach

From 1988 to 1999, Mr. Werner gained experience in the area of SAP R/2 with the main focus on SAP basis, logistics, and programming (ABAP and Assembler). As a SAP R/3 application consultant for the modules MM, WM, and PM, he has carried out ABAP developments for ADD ONs, interfaces, and enhancements since 1999.

Andreas Wiegenstein, Managing Director and Chief Technology Officer (CTO), Virtual Forge GmbH

Since 2002, Mr. Wiegenstein has been working in the area of SAP security. As a co-author of the book "Secure ABAP programming" (SAP Press), he frequently lectures on the subject of SAP/ABAP security and compliance at international conferences, such as RSA, Black-Hat, Hack In The Box, Troopers, SAP TechEd, etc. and DSAG events.

In addition, the following persons have contributed significantly to the up-to-dateness of these guidelines based on the provision of documents, review activities, and numerous discussions:

Michael Cendon, Thorsten Franz, Pascal Mannherz, Thomas Prang, Markus Tradt, Tobias Trapp, Peter Weigel, Marc Zimmek

Special thanks to SAP, in particular to Horst Keller and Dr. Wolfgang Weiss, who supported the work on these guidelines with constructive proposals and reviews.

The translation of this document from German into English was done by the following:

Charles Currey, Dr. Christian Drumm, Jürgen Wachter, Andreas Wiegenstein, Bärbel Winkler

10 APPENDIX: NAMING CONVENTIONS

Below you find an example on how to build up ABAP development naming conventions. This example can be used as a reference for creating own naming conventions and has to be adpated to the industry and company requirements.

General hint: Objects in ABAP dictionary have different limits for the number of available characters. This should be considered, when naming objects.

In our case the customer namespace Y... will be used. Customer namespace Z... or customer namespace /.../ could be used instead, like described in chapter 2.

Abbreviation

Abbreviation	Meaning
Mm	Modul or Project Abbreviation mm represents a SAP module (i.e. PP MM) or customer specific project.
Uu	Application area An application area optionally allows a more exact classification of module or project. The application area is defined by the project manager.
K	constant
C	Alphanumeric character
N	Numeric character
...	Characters of any length

10.1 GENERAL NAMENING CONVENTIONS

Structured elements

Type	Convention	Example
Package	Ymmuu...	YPPRUECK
Reports	Ymmuu...	Ymmuu...
Modul Pools	SAPMYuu	SAPMYCAUB
Includes	TYmmuuccc..._(A)*	MYCAUB_TOP
Transactions	Ymm..	YMM01
Message Class	Ymm..	YPPRUECK
WebDynpro ABAP	Ymm...**	YPKS_ADMIN_1

Remark:
*Include starts with the name of the module pool without prefix "SAP".Type of include is specified at the end:

- TOP Top Include
- PBO Process Before Output Include
- PAI Process After Input Include
- FORMS Include for forms
- CLASS Include for local Classes

** The usage of WebDynpro editors (i.e. in se80) results in an automatic creation of source code. This complicates the feasability of own naming conventions.

Data-Dictionary-Objekte

Type	Convention	Example
Tables	Ymmuu...(t)	YPPORDER
Views	Ymmuu..._V	YPPORDER_V
Table types	Ymmuu..._TT	YCADOCUMENT_TT
Structures	Ymmuu..._S	YCADOCUMENT_S
Data elements and Domains	Ymmuu...	YCAFTDATUV
Search help	Ymmuu....	YPPPROJE
Lock objects	EY_<Tabelle>	EY_YSDMINFO
Type groups	Ymmcc	YCA02

Classes & Interfaces

Type	Convention	Example
Classes	YCL_mm_...	YCL_SD_FAKTURA
Interfaces	YIF_mm..	YIF_SD_BOOKING

10 APPENDIX: NAMING CONVENTIONS

10.2 ATTRIBUTES

Locale class variables and attributes should be declared as private. Attribute access has to take place by get and set methods.

LV_... Attribute variable (local value ...)

LS_... Attribute structure (local structure ...)

LT_... Attribute table (local table...)

10.3 METHODS

Methods should have a short, self-explanatory, English name. The following prefixes describe the functionallity of the method:

set_... Set values

get_... Get values

send_... Send information

save_... Save data to database

etc.

The exact functionallity should be placed into the description of the method. If there is not enough space (60 characters), there should be a detailed description of the method in the class documentation.

10.4 SIGNATURE OF METHODS

Method parameters should be named as described below:

Nr.	Parameter type	Prefix
0	Importing	I_...
1	Exporting	E_...
2	Changing	C_...
3	Returning	R_...

10.5 FUNCTION GROUPS AND MODULES

Type	Convention	Example
Function Group	Ymm_ccc_...	YCA_KONFIGURATION
Function Module	Ymm_...	YSD_FAKTURA_ZU_WE

Function module interfaces are similar to the interfaces in object oriented programming (see 10.4).

BEST PRACTICE: Tables can be transfered as import, export or changing parameters, so that it is clear, which tables are changed in the function modules and which ones not. ABAP syntax check ensures that importing parameters are not changed.

10.6 ENHANCEMENTS

Type	Convention	Example
Enhancement Spot	YES_...	YES_MV45A
Enhancement Definition	YED_...	YED_ MV45A
Enhancement Implementation	YEI_...	YEI_ MV45A

10.7 FORMS

Type	Convention	Example
Adobe Forms – Formular	Ymm..	YPP_VBK1
Adobe Forms – Interface	Ymm.._IF	YPP_VKB_IF
Smart Forms/SapScript – Forms	Ymm..	YPP_VBK1
Smart Forms/SapScript – Style	Ymm..	YPP_VBK1
Smart Forms/SapScript – Text modules	Ymm..	YPP_VBK1
Searchhelps	Ymmuu....	YPPPROJE
Lock Objects	EY_<Tabelle>	EY_YSDMINFO
Type groups	Ymmcc	YCA02

10.8 JOBS

Type	Convention	Example
Jobs	YMM_<Prio>_<BUKRS>_<DESC>	YSD_1_1000_

<Prio> = Priority, single number, <BUKRS> = Company code

10 APPENDIX: NAMING CONVENTIONS

10.9 DATA ELEMENTS

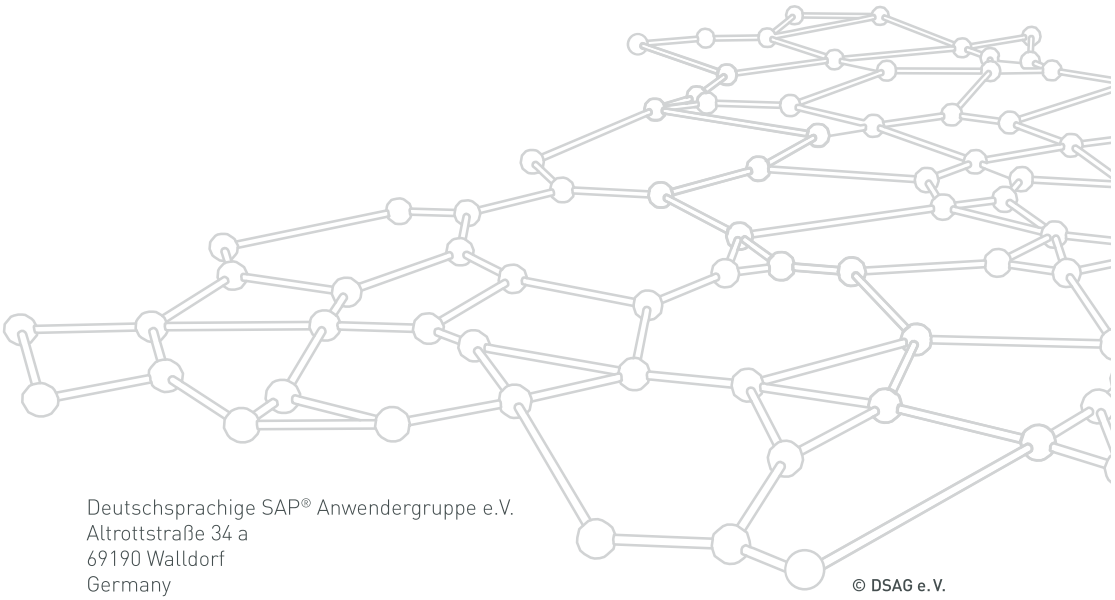
Data type	Prefix Part
Elementay Type/Variable	v
Structures	s
Tables	t
Data Reference	r
Class Reference	o
Interface Reference	i
BADI Reference	b
Exception Class Reference	x

This prefix parts build the type depending component of the context depending prefix shown below. [t] has to be replaced by the suitable type depending prefix part

Type of Declaration	Naming convention
Local Variable	l[t]_*
Global Variable	g[t]_*
Static Variable	s[t]_*
Local Field Symbol	<l[t]_*>
Global Field Symbol	<g[t]_*>
Local Constant	lc[t]_*
Global Constant	gc[t]_*
Select-Option	s_*
Parameter	p_
Function Module Parameter	i[t]_* für Importing
e[t]_* for Exporting	s[t]_*
c[t]_* for Changing	<l[t]_*>
t[t]_* for Tables	<g[t]_*>
FORM Parameter	p[t]_* für Using
c[t]_* for Changing	l[t]_*
t[t]_* for Tables	g[t]_*
Table type	tt_*
Structure Type	t_*

Objectoriented Programming:

Entity	Namening Convention
Local Class	lcl_*
Global Class	cl_*
Local Interface	lif_*
Global Interface	if_*
Instance attribute	m[t]_*
Static Attribute	g[t]_*
Constant	c[t]_*
Method Parameter	i[t]_* for Importing
e[t]_* for Exporting	p_
c[t]_* for Changing	i[t]_* for Importing
r[t]_* for Returning	s[t]_*
Event Parameter	i[t]_*



Deutschsprachige SAP® Anwendergruppe e.V.
Altrottstraße 34 a
69190 Walldorf
Germany

Phone: +49 (0) 62 27 – 358 09 58
Fax: +49 (0) 62 27 – 358 09 59
E-Mail: info@dsag.de
Homepage: www.dsag.de