General guidelines for

SAP development

Version	2.5
Status	released
Date	21.12.2016
Valid from	01.01.2017
Final version:	

Authors:

abat QS Team (sq@abat.de)

Change History:

V 2.1	TBA	Safety section completed
		Logo changed
V 2.2	TBA	Safety section adjusted with VW guidelines
V 2.3	MAG	Modifications, modularization, language skills
	JOM, ROL	Naming convention
V 2.4	MAG, ROL	Text translation (internal development)
V 2.5	BEN	Revision and update

Content

Scope of application

These guidelines apply to developments carried out by abat AG on behalf of customers on customer systems as well as for developments in own abat systems. They are the basis of internal quality assurance and will be agreed with the customer.

Any customer- or project-specific requirements which deviate from this directive can be agreed and will then replace the relevant sections of these guidelines or apply in addition to this policy. However, for checks and reviews the stricter individual directive (customer or abat) applies.

1 General guidelines for SAP developments

1.1 Namespace concept

The namespace rules of the respective project (customer name space / project namespace, general naming conventions for customer developments) apply to all development objects.

1.2 Developer Key

Developers will be registered centrally for each project by the competent authority and the developer key will be requested from there via the OSS.

Project-specific regulations have to be considered.

1.3 Modifications

Any modification as change of foreign namespace objects (SAP or 3rd party namespace) are to be avoided. As modifications are evaluated written accesses to SAP standard tables with customer-specific coding, written accesses to non-programmable variables of the standard from customer coding (dirty assign) and changes of software in namespaces for which no development / repair license is available in the system.

In any case, check if the component can be copied into the customer namespace, or if a modification can be avoided by using other options.

If a modification is inevitable, the following rules apply for the implementation if there are no other agreements in the project:

- Modifications must be approved by the customer

- Modifications must be implemented by using the modification assistant.

1.4 Code reviews

To ensure the quality and compliance with the development guidelines and the requirements for standards compliance, code reviews will be carried out on a regular basis. This process is conducted in coordination with the respective project management.

The results will be documented for each development object. All necessary corrective actions will be monitored by the reviewer.

The results of the reviews will be provided to the respective project management.

1.5 Scope of development tasks

The development of an object will be considered complete when the following criteria are met:

1. The software was developed based on the functional specification.

2. The design of the software has been quality-assured under 4-eyes principle (per the agreed scope of testing) before the development beginning.

3. The software complies with the development guidelines in the current version.

4. A developer test was performed and documented.

5. A technical documentation has been created.

6. Depending on the release, the development must be checked with the code inspector (SCII) or ABAP test cockpit (ATC) in the respectively agreed test variant. Errors must be corrected and warnings should be checked.

1.6 Modularization and packaging of functional units

All functional units / objects / libraries must be kept free of dependencies, it means that each development should be encapsulated as far as possible so that the migration into other systems as well as reusing them in other developments within the same system can be done with minimal effort.

Within the program flow, the functional and technical units should be clearly separated.

As a minimum requirement is considered the separation of:

Surface / data collection / processing logic / update functions.

In the design of modular units, the preference sequence is as follows:

1. ABAP Objects
2. Function modules
3. Reports
4. Form routines (form NW 7.3 obsolete)

If function blocks or subroutines should be used for technical reasons, the functionality is to be linked into local classes.

1.7 Standard Compliance

Developments are to be implemented so that they do not cause conflicts with the standard SAP software i.e. standard functions mustn't be impaired or altered. The consistency and integrity of the standard data model must be guaranteed.

The same applies to third party software (partner products).

1.8 Internationalization

In monolingual projects the option of multi-language support has to be taken into account by using and respecting the mechanisms envisaged to achieve language independence. In case of multilingual projects, an original language must be defined from the beginning.

No language-dependent elements must be defined directly in the source code. Text elements should be used instead.

The following rules apply to ensure multi-language support:

1. Clear definition of the relevant logon languages per system.
2. No text within the source code (literals)

   Note:

   Text literals as a text symbol should be avoided.

```
" literal textsymbol
gv_string = 'Mein Text.'(001).

" besser:
gv_string = text-001.
```

3. For data elements, text symbols, etc. the field length must be at least 25-50% longer than the original. The actual length should be based on the expected length of the translation and should be asked in case of doubt or set up accordingly. A suitable place must be provided on surfaces.

4. Avoid specific or uncommon abbreviations in the original language.

5. Proper use of text symbols (no texts which are concatenated from several text elements, since the syntax in other languages can be different).

6. Instead of anonymous placeholders (&), unique placeholders (&1, &2, etc.) are to be used in message texts.

In case of internal development (abat) all texts must be written in German and English.

If SET LOCALE is used, the logon language must be restored.

Only 7-bit ASCII characters should be used in the source text.

If the texts contain composite characters or surrogates, care must be taken that these are not separated during the cutting of the text. More information can be found here (http://help.sap.com/abapdocu_750/en/abensplit_text_guidl.htm).

Text files are to be read and described via the addition ENCODING in UTF-8 and via the addition BYTE-ORDERMARK.

2 Guidelines for ABAP development

2.1 Naming Conventions

All development objects are subject to strict naming conventions. Like comments, objects are to be named in English. In addition, the names should be chosen as speaking as possible.

If no customer specific conventions exist, the samples below are applicable.

All development objects must be developed within the customer namespace.

Exceptions are provided by SAP predefined namespaces like for:

1. Customer-includes in tables
2. Include - Programs in Customer Exits
3. Appends to search helps
4. Authorization Objects

Program internal objects such as parameters, global or local variables are also subject to naming conventions.

If project-specific naming conventions exist, the rules for naming must at least meet the following criteria:

- Identification of the scope of the object (global / local)
- Identification of the type of the object (parameter, variable, structure, internal table…)

2.2 General Guidelines for ABAP Development

2.2.1 Code Documentation

Each development must be documented:

- External documentation of development in an appropriate format and/or online documentation of development within the system
- Inline documentation of the source text
- Separate security documentation, if required

Inline documentation / comments should be present at all required locations in the code. Coherent processing blocks should be commented at their beginning. The same applies to blocks where the sense is not obvious at first sight.

Where error handling differs from the specifications because there may be no business need, as well as code fragments that do not comply with the safety guidelines, an explanatory comment should be introduced by and mentioned in the approved safety documentation.

The commentary language is English, if the customer has no other requirement.

Comments must always be of serious content.

A comment should not describe how something is done, but why.

Performance critical programs may additionally require a runtime trace (ST12) in the documentation which is carried out with a representative or scalable data volume.

Additional documentation requirements may apply for each Customer /Project.

The online documentation should include at least the following content:

With dialog transactions:

Explanatory text to assist the user and a description of the functionality in a way that the documentation enables the user to carry out at least the essential functions of the software without further explanations.

In general:

Explanatory Content for support staff / AMS, which describes the basic functionality, sketches the program sequence and thus allows an introduction to the application.

2.2.2 Change Management and Documentation

Each program must have a header that includes the following information:

- Functional brief description
- Dependencies
- Name of the functional managers
- Name of developer
- Version history (if an object is migrated to other systems and thus or as a result of system copies SAP Versioning gets lost) with a short description of the change.

All program changes must be documented in the source code by change markers. These markers have to be indicated in the version history, making the search for code modifications easier.

2.2.3 Code Clearing

Commented code that is no longer used should be removed from code from time to time on request so that the program is kept clear and possible version comparisons do not consist mainly of the differences between the commentaries.

The supply of further systems from an existing landscape should always be made with ‘clean’ coding.

The initial transport into a productive environment should be made with clean coding.

2.2.4 ABAP – Standards

ABAP for the syntax bases to be used in ABAP programs, the latest ABAP version (release-dependent) is applicable with the restrictions, that means:

Programs must always comply with ABAP Objects (OO) syntax restrictions, statements which are not permitted in the OO context are also not permitted in the 'simple' context (reports, FB). Also, obsolete statements cannot be used.

Even in non-Unicode systems all programs must comply with Unicode restrictions.

2.2.5 Serviceability

In terms of serviceability the following guidelines must be observed:

- There must be no editor locks
- The source code must not be hidden unless this is required by the customer. For these exceptional cases, the necessity has to be explained in the specification and must be approved by the development management in the project as well as documented separately. In any case, serviceability of software must be ensured.
- The SAP usage statement is to be supported as far as possible. Particularly in the case of dynamic calls, it must be ensured that usage points can be recognized.
- Constants or customizing tables should be used instead of writing values into the source code (magic numbers).

2.2.6 Authorization Checks

All dialog or externally usable functionalities (reports, dialog transactions, RFC modules) must contain mechanisms for authorization check. The default implementation of the authorization check must be used. Pseudo checks such as via hard coded user name must not be implemented.

The authorization concept is part of the functional specification and lies within the responsibility of the author of the specification.

The lack of authorization checks must be justified and documented in the technical documentation.

2.2.7 Data Declarations

Data declarations must correspond to the naming conventions for program-internal objects. They always belong to the beginning of the respective code block. Declarations within functional code blocks are not allowed (except for inline declaration from NW 7.4 onwards). Generally speaking, variable names should be used.

Local data types, interfaces, and classes of a class pool are to be created as class-local types. If such types are used in methods of the PRIVATE SECTION, they must be defined by the Local Class Definitions / Types Function, so that friendly classes can access them.

The use of untyped variables (especially as formal parameters in form routines) is not permissible. The exception is generic field symbols that are dynamically typed if different types are required at runtime. Otherwise at least the type 'any' or 'any table' is to be used.

Required data types should be defined in the dictionary or in the TYPES statement, rather than declared as a bound data type in the DATA instruction. The declaration should take place in the context which ensures the best possible encapsulation.

Dependent data objects (e.g. assigned field symbols during iterating via tables depending on the type of the table) should be typed via LIKE. In all other cases the typing over TYPE is to be used.

When declaring a structure, components of other structures are not to be integrated via INCLUDE, but are to be incorporated into a real substructure.

The addition VALUE allows only values of the type, content, and length which exactly match to the data type of the declared data object.

2.2.8 Implementation

Procedures must be left with RETURN (instead of EXIT or CHECK).

No implementation is to take place in dialog modules or event blocks.

2.2.9 Macros

The use of macros is allowed, but subject to the following restrictions:

- No database operations in macros
- No complex logic in macros

2.2.10 Access to System Fields

Only read access is allowed for system fields (sy-xxx, fields of structure SYST) from programs. Write accesses occur through the runtime environment. Obsolete or internal system fields should not be used. The evaluation of system fields is to be carried out only directly after instructions, for which it is documented that they set the appropriate fields. The return value sy-subrc is to be evaluated where necessary.

If system fields are used as the actual parameters for a procedure, the value is to be copied into an auxiliary variable, which is passed. A reference transfer of a system field can lead to unexpected behavior. Since a value transfer can be changed to reference transfer unnoted, the intermediate step must be taken via the auxiliary variable.

System fields are not intended to be used on user interfaces, and in no case, in statements that they make.

2.2.11 Security / Compliance

If deviations from the following rules are necessary for business reasons, these must be agreed separately and documented.

2.2.11.1 Authorizations

In principle, the software must always be designed in such a way that no user can gain access to data contents for which his or her own authorizations are not sufficient. Therefore, permissions are always to be checked during access and presentation.

Each executable program and each RFC-enabled function module must have an authority check.

Programs must always be executed with the authorization of the current user.

CALL TRANSACTION is permitted only if there is no other option.

If Call Transaction is used, make sure there is an authorization check for the target transaction. The reason for using Call Transaction should be noted as comment within the source code. From NW 7.4 CALL TRANSACTION is obsolete and is replaced by CALL TRANSACTION ... WITH AUTHORITY-CHECK...

AUTHORITY-CHECKs should always be inserted via pattern / auto-completion. Unused fields must be provided with the value DUMMY. The sy-subrc must be handled.

Any access to application data on the server whose selection is dependent on a user input must have an authorization check.

Used paths should be maintained only via transaction FILE and determined using their function blocks (see also https://blogs.sap.com/2013/08/05/protecting-abap-code-against-directory-traversal-attacks/).

2.2.11.2 Dynamic Programming Techniques

Dynamic programming techniques must always be used carefully. If they are used, all possible errors are to be excluded or intercepted, as otherwise run-time errors may occur. Memory bottlenecks can also occur easily. How to prevent this can be found in the ABAP Programming Guidelines for SAP (http://help.sap.com/abapdocu_750/en/abenmem_cons_dyn_mem_obj_guidl.htm).

Dynamic SQL statements are allowed only if there is no other possible solution. In any case it must be ensured that the user does not obtain access to data for which he does not have authorization. In particular, the possibility of an SQL injection has to be considered. For the verification of the dynamic WHERE clause, SAP provides the class "CL_ABAP_DYN_PRG". Here (http://scn.sap.com/community/abap/blog/2013/11/20/how-to-protect-your-abap-code-against-sql-injection-attacks) is described how using this class also SQL injections can be avoided.

No dynamic generation or programmatic manipulation of code may occur. In most cases the RTTI / RTTS classes can be used for the solution.

2.2.11.3 Input Validation

User inputs must be validated generally and in the backend. Programs for SAP GUI can also be compromised by manipulating the client. If possible, a whitelist filter should be used.

When accessing files from the application server, for example, through the commands (OPEN | DELETE | TRANSFER | READ) DATASET, a directory traversal attack is possible. Such an attack should be avoided by filtering the relevant characters (.: \ /), the selection of files from an index or best by the complete independence of user input.

The option FILTER of OPEN DATASET allows operating system commands to be executed on the application server. It is not possible to enter user-defined values for this field.

RFC-enabled function blocks must handle and validate all incoming actual parameters such as user inputs.

2.2.11.4 Diverse

Iterative or recursive statement blocks, in which the number of passages can be influenced from outside (users), or where the number of iterations depends on the quality of the data to be processed, shall be limited in the program to a fixed maximum number of iterations or to be provided with a termination condition in order to avoid endless loops.

All application data must be stored exclusively on the server.

Encoded breakpoints must be removed before transport to a productive system.

Test code (additional expenses or similar) must be removed before transport.

No direct C calls / kernel calls.

System commands should not be executed. If this is unavoidable, the call is to be performed via constants.

No editor call.

No programmatic manipulation of user master data or permissions

No direct specification of filenames on the server (use transaction FILE).

No use of task handler functions (FuGr THFB).

No call for external form routines.

All functionalities that operate on the server or client at the operating system level or file system level, or OLE functions on the client must be documented separately.

Accesses to the user front end are to be designed in such a way that all operations are presented to the user completely and transparently.

No programs that delete, create, modify or show source code to not-authorized users.

No change of coding while circumventing the planned transport route.

Native SQL must not be used.

The program sequence must not depend from:

- System
- Client
- User
- Operating system
- Database version
- Host
- Date

2.2.12 Error Handling

Any possible error which can occur must be handled appropriately.

This applies to explicit errors such as exceptions in methods or function modules, as well as to implicit errors such as division by zero, errors by opening files, unassigned field-symbols etc.

For each development, it is necessary to ensure that the occurrence of errors is handled appropriately and that occurring errors are issued depending on the type of development (messages for dialog applications, application log or spool for interfaces).

Messages are maintained for a message class in transaction SE91. It is important to ensure that the text of the message is meaningful. In principle, a long text must always be created for each message, and the long text can only be omitted in exceptional cases, for example, if the full meaning is clear from the message text. However, messages should be used only for error handling on classic screens or as a storage for exception texts.

A reference to the respective business object must be created via the message variables. For example, error messages to a list of purchase order items must be assigned a unique reference to the respective purchase order item.

Messages which are sent dynamically must be able to be found using the usage statement.

Exceptions in function blocks or methods are to be implemented exclusively by class-based exceptions.

Appropriate attributes ensure that a caller can evaluate the corresponding information. Exception texts are to be created in message classes and only these are used. If a meaningful program execution is no longer possible, the processing is not to be terminated via a message of type 'X', but via an assertion.

ABAP statements that contain intermittent runtime errors due to exception classes must be treated or forwarded appropriately.

2.2.13 Web Programming

The development of own HTTP handlers is to be refused for security reasons. Instead, the standard SAP UI frameworks can be used.

The Whitelist filter of class CL_HTTP_UTILITY is to be used.

To prevent cross-site scripting attacks, user data must be encoded before output. Class CL_HTTP_UTILITY provides methods for this.

The start of a Web Dynpro application is not intended to directly trigger a business process.

2.3 Type Conversions

Type conversions should be avoided as far as possible. The use of literals must be type-compliant. The frequent conversion between sliding and fixed point numbers leads to large rounding errors.

Unexpected conversion results should be avoided by using MOVE EXACT.

2.4 Encapsulation / Reusability

Program code should be written in such a way as to facilitate reuse in another context. To achieve this, important programming principles are to be observed such as the separation of concerns, open / closed and a loose coupling.

2.5 Reports

For the design of executable programs (reports), the following rules apply:

1. Each report must have a selection screen. For the selection screen, the principles listed under 'Surfaces' apply. In addition, validations and input aids are to be implemented for the parameters / elements.

2. Reports should be split into suitable includes (TOP include, selection screen, form routines).

3. Includes must not be used more than once.

3. The standard event blocks should be implemented as far as this is useful (minimum: *START-OF-SELECTION*).

2.6 Function Groups / Modules

Functional groups should be kept small.

Generated function modules (e.g. table maintenance or enqueue modules) shall be clearly separated from hand-coded function blocks (separate function group).

Only function modules which work directly together (e.g. share global data), or those that are always called within the same context should be implemented in the same function group.

The global data areas must be restricted to the most necessary. RFC-capable modules, which are intended to be used in the aRFC parallelization, do not use global data areas because write access to global data in parallel processes can have unpredictable side effects.

2.7 Dialog Applications

2.7.1 Ergonomics and Design Guidelines

The design examples of SAP should be followed, see the SAP Library under Basis → ABAP Workbench → BC SAP Style Guide (transaction BIBS).

When selecting the interface elements (depending on the release), the respective current front-end controls are preferred.

The ISO Dialogue Principles (ISO-Standard 9241-10) should be considered as far as possible.

2.7.2 Accessibility

To ensure accessibility, the following points must be ensured:

- All input and output fields have meaningful names
- All table columns have a heading
- Each icon has a Quick info text
- Information is not displayed exclusively by colors
- Input and output fields are grouped in frames with meaningful titles

2.8 Batch Programs and Interface Reports

Interfaces must be designed in such a way that continuation is ensured at any time during the runtime without data loss or inconsistency.

Batch programs and interfaces must have log functions that make the status and especially the errors of each run visible in appropriate form.

2.9 ABAP Objects

ABAP Objects should be used for all new developments.

The principle of separation of interests (SoC) is to be respected.

2.10 Extension Concepts and Implementations

2.10.1 Field-Exits

Since field exits have not been supported since release 4.6, as well as because of the known problems with regard to performance and stability, field exits are no longer permitted.

2.10.2 BAdI

When implementing BAdIs, the respective release status of the basic system must be considered. From 7.0, the new implementation concept should be preferred.

2.10.3 Implicit Enhancements

In principle, implicit enhancements have the same effect as modifications. It is therefore important to ensure that the functionality and the consistency of the data in the surrounding standard software are not damaged.

2.10.4 Implementation of SAP Notes

For the installation of OSS messages, the Note Assistant (transaction SNOTE) must be used as far as possible.

2.11 Persistent Data

2.11.1 Reading of Data

The data of other clients should not be read.

For all readings, an appropriate performance must be considered and the runtime behavior has to be validated and documented using SQL trace.

2.11.2 Accounting

The accounting logic of the standard is to be used for the accounting of data, that is, updateable function blocks called up in UPDATE TASK.

Exceptions are permissible only for special cases (immediate reading of the data, etc.).

No further application logic is implemented in accounting blocks. The error output is made using simple MESSAGE eXXX (XX) instructions, so that the errors in the accounting cancellation can be evaluated.

The use of 'MODIFY' is prohibited for database operations.

2.11.3 Locking

The SAP locking mechanism must be used, whenever data is written to the database.

2.11.4 Performance

Every development should assume of a maximum dataload and must be designed accordingly. Database accesses and business logic should be implemented with respect to the performance.

2.11.5 Maintaining of Database Indices

Database indexes must be maintained and managed centrally for each SAP installation. The corresponding regulation is the responsibility of the project, but should consider the guidelines of the SAP.

2.11.6 Accessing SAP Standard Database Tables

If write access to SAP standard data is required, the SAP standard posting modules or BAPI’s must be used.

A write access with 'own' coding on standard tables is interpreted as a modification.

However, if such access is inevitable, the following rules apply:

- Consistent posting i.e. all dependencies are taken into account.
- Standard locking must be used

- Logging the changes, so that the search for errors is facilitated in case of inconsistencies.

3 Appendix

3.1 Name Conventions for Dictionary Objects (Examples)

Databank tables	/name space/name
Data elements	/name space/name
Domains	/name space/name
Views	/name space/v_name
Structure	/name space/s_name
Table type (standard)	/name space/t_name
Table type (Sorted)	/name space/ts_name
Table type (Hashed)	/name space/th_name

In the table description, mention the key, if possible.

Append Structure	/name space/s_name

In the table description, mention the key, if possible.

Append structure	/name space/s_name

3.2 Name Conventions for Repository Objects

3.2.1 Programming

Development class	/name space/name
Report	/name space/name
Transaction code	/name space/name
Function group	/name space/name
Functional module	/name space/name

Class	/name space/cl_name
Exception class	/name space/cx_name
Interface	/name space/if_name
Method	name

Use speech-based method names to deduce what is done in the method.

Examples of method names:

Query to Boolean values	IS_NAME
Set variables	SET_NAME
Read variables	GET_NAME

3.3 Name Conventions for Program-Internal Objects (Examples)

Constants and types are always to be defined globally, so there are no conventions for local constants / types. Instance and class attributes count as global data.

3.3.1 Internal Data Objects

Internal data objects and formal parameters must be prefixed. This prefix is composed of a letter for specifying the context (k) and one for specifying the object type (y). The prefix should be separated from the actual name with a dash. An identifier thus looks as follows:

[Ky] _name

Where k and y are replaced by the respective letters. Here are specific examples:

Global data object of elementary type:	gv_name
Table as changing parameter:	ct_name.

For field symbols, the mandatory pointed brackets must be grouped around the name. Select-Options, Parameters, and Constants will be named not after this scheme.

3.3.1.1 Context-dependent Prefix Component

Context	Prefix component

Internal program type	t
Global data object	g
Lokal data object	l
Importing-parameter	i
Exporting-parameter	e
Changing-parameter	c
Returning-parameter	r
Using-parameter	u

3.3.1.2 Type-dependent Prefix Component

Data type	Prefix component
Elementary type	v
Structure	s
Table	t
Range	r
Object reference	x
Data reference	z

3.3.1.3 Select-options and Parameters

Select-Option	s_name
Parameter	p_name

3.3.1.4 Constants

Constant data field	cv_name
Constant structure	cs_name