SDN Community Contribution

(This is not an official SAP document.)

Obsolete ABAP Language Constructs

SAP DEVELOPER NETWORK

Applies To:

4.6C and above

Summary

A number of years ago, while navigating the ABAP help and new release notes, I found a list of all the ABAP Language constructs that were restricted with the advent of ABAP Objects. It was explained that as part of a clean-up of the ABAP language for ABAP Objects, stricter syntax checks in ABAP Objects are performed for constructs previously allowed in ABAP. Some statements are entirely obsolete in the context of ABAP Objects, both in the declarative statements of Class Definitions, and in the coding of methods in the Class Implementations. The following sections contain obsolete syntax for ABAP Objects and show the language constructs by which the obsolete syntax can be replaced.

Although you are generally not forced to modify your existing programs to represent these changes, it would be a very good practice to implement these rules and constructs with any new code development and coding samples.

By: Marilyn Pratt

Company: SAP Labs

Date: 11 Nov 2005

Table of Contents

Incorrect Syntax

Missing Separators

Separators (blank character, comma, colon, period or end of line) are required in ABAP Objects after literals and offset/length specifications.

Error message in ABAP Objects if the following syntax is used:

```
CONCATENATE 'fgfdg'f INTO g.

WRITE AT /off(len)'...'.
```

Correct syntax:

```
CONCATENATE 'fgfdg' f INTO g.

WRITE AT /off(len) '...'.
```

Reason:

Standardization of the syntax of statements. It is not allowed to write the next statement (token) immediately after any part of a statement; a valid separator must always be inserted.

Incorrect Plus-Parentheses Notation

Empty plus-parentheses notations are not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
DATA: f1() TYPE ...,

f2+  TYPE ...,

f3   LIKE f1+().

SELECT SINGLE ... FROM +(f1) INTO (f2+off(), f3+(len)).

WRITE AT +(len) f3().
```

Correct syntax:

```
DATA: f1 TYPE ...,

f2 TYPE ...,

f3 LIKE f1.

SELECT SINGLE ... FROM (f1) INTO (f2+off, f3(len)).

WRITE AT (len) f3.
```

Reason:

Outside of arithmetical expressions, plus signs can only be used for offset/length specifications in field names. Plus signs are obsolete if they are not followed by an offset value. The system currently interprets a single plus sign following a field name or a plus sign directly followed by a parenthetical expression as non-existent. Therefore plus signs can also be used when no offset/length specification is involved, for example, in data declarations where only length specifications are possible and before dynamic expressions. An empty parenthetical expression after a plus sign, an offset value or a field name is interpreted as non-existent and is obsolete, too.

No literals over several lines

In ABAP Objects literals are not allowed to extend over several program lines.

Error message in ABAP Objects if the following syntax is used:

```
WRITE 'Start...

                      ...end'.
```

Correct syntax:

```
WRITE 'Start...     ' &

'        ...end'.
```

Reason:

The number of blank characters inserted depends on the line length of the editor. The line length of the editor is no fixed value but can be increased in a later release. For literals that are longer than one editor line, the & character can be used to combine several literals.

No continuation of field names beyond the end of the line

In ABAP Objects it is not allowed that field names continue beyond the end of the line in listings that are enclosed in brackets.

Error message in ABAP Objects if the following syntax is used:

```
SELECT SINGLE col1 col2 ... coln

FROM dbtab

INTO (wa-col1, wa-col2, ................ , wa-c

          oln)

           WHERE col1 IN (f1, f2, .................... , f

          n).
```

Correct syntax:

```
SELECT SINGLE col1 col2 ... coln

FROM dbtab

         INTO (wa-col1, wa-col2, ................ ,

             wa-coln)

        WHERE col1 IN (f1, f2, .................... ,

                 fn).
```

Reason:

Field names must never extend over several lines. If the editor line length is increased in a future release, field names that are divided by the end of a line will cause syntax errors. The behavior in listings is an exception and will be adjusted to the general handling.

Type Definitions and Data Declarations

New naming conventions

The names of components in classes, that is attributes, methods and events, must consist of the characters "A-Z", "0-9" and "_". They must not begin with a number.

Error message in ABAP Objects if the following syntax is used:

```
DATA: field-name TYPE ...,
                1name TYPE ...
```

Correct syntax:

```
DATA: field_name TYPE ...,
name1 TYPE ...
```

Reason:

Special characters should not be used in names because they often have a special meaning. The new naming conventions correspond to the conventions of other programming languages.

TABLES statement not allowed

Creating table work areas with the TABLES statement is forbidden in ABAP Objects.

In ABAP Objects, an error message occurs on:

```
TABLES dbtab.
```

Correct syntax:

```
 DATA wa TYPE dbtab.
```

Reason:

The semantics of the TABLES statement is ambiguous. Instead of table work areas, you should use explicit work areas. Common interface work areas for passing data between programs and procedures are not supported in ABAP Objects. Only the public components of a class can be used as its interface, that is, its public attributes, and the interface parameters of methods and events.

Data transport between ABAP programs and logical databases or screens using global table work areas is not supported in ABAP Objects, and has been replaced by other techniques.

Statement NODES not allowed

In ABAP Objects it is not allowed to create node work areas using the statement NODES .

Error message in ABAP Objects if the following syntax is used:

```
NODES struc.
```

Correct syntax:

```
 DATA wa TYPE struc.
```

Reason:

The previous way of processing logical databases is not supported by ABAP Objects. Data transports between ABAP programs and logical databases using global work areas are not provided for in ABAP Objects.

No common data areas

It is not allowed in ABAP Objects to create common data areas using the addition COMMON PART of the statement DATA is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
DATA BEGIN OF COMMON PART c,

...

DATA END   OF COMMON PART.
```

Reason:

Common interface work areas for data transfer between programs and procedures are not supported in ABAP Objects. In classes only the components visible to a user are used as interfaces, i.e., visible attributes and the interface parameters of methods and events.

No LIKE reference to Dictionary types

In classes only the TYPE reference can be used to refer to data types in the ABAP Dictionary. The LIKE reference is allowed to local data objects only. In local classes this includes the attributes of the class and the data objects of the main program. In global classes only the class attributes can be referenced. This applies both to data declarations and to type assignments of interface parameters and field symbols.

Error message in ABAP Objects if the following syntax is used:

```
DATA f LIKE dbtab.
```

Correct syntax:

```
 DATA f TYPE dbtab.
```

Reason:

The TYPE addition is designed to be the only construct that enables references to data types, whereas the LIKE addition is used only for data objects. The repository objects in the ABAP Dictionary are data types but not data objects. Outside of ABAP Objects the LIKE reference to database tables and flat structures in the ABAP Dictionary is still allowed for reasons of compatibility with previous releases.

No implicit specification of type, length, and decimal places

In the statement TYPES, the type must be specified explicitly with type C, the length with types C, N, P, and X, and the number of decimal places with type P in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
TYPES: t1,

t2 TYPE p.
```

Correct syntax:

```
TYPES: t1(1) TYPE c,

t2(8) TYPE p DECIMALS 0.
```

Reason:

The complete type definition is required in order that types with an incomplete definition can later be regarded as generic types. In the statement DATA, short forms are still completed.

Incorrect length specification in declaration

Length specifications are not allowed in ABAP Objects in connection with type assignments of data type D, F, I, T in the statements TYPES , DATA, CLASS-DATA, STATICS and CONSTANTS.

Error message in ABAP Objects if the following syntax is used:

```
DATA: f1(8) TYPE d, f2(4) TYPE i.
```

Correct syntax:

```
DATA: f1 TYPE d, f2 TYPE i.
```

Reason:

The built-in elementary types D, F, I and T have fixed unchangeable lengths. Other length specifications are not allowed anyway. Specification of the predefined lengths is obsolete.

Operational Statements Not Allowed in Structure Definitions

Within the definition of a structured data type or object using TYPES, DATA, CLASS-DATA, STATICS, or CONSTANTS, no other statements can occur within ABAP Objects.

In ABAP Objects, an error message occurs on:

```
TYPES: BEGIN OF line,

           col1 TYPE i.

  MOVE 'X' TO a.

  TYPES:   col2 TYPE i,

         END OF line.
```

Correct syntax:

```
TYPES: BEGIN OF line,

           col1 TYPE i,

           col2 TYPE i,

         END OF line.
```

MOVE 'X' TO a.

Reason:

The definition of a structure between BEGIN OF und END OF is a closed unit in which you can only declare components of the structure.

Anonymous Components of Structures Not Allowed

Within the definition of a structured data object using DATA, CLASS-DATA, STATICS, or CONSTANTS, you cannot declare any anonymous components in ABAP Objects.

In ABAP Objects, an error message occurs on:

```
        DATA: BEGIN OF struc,
              'Text Literal',
              space(10) [TYPE c],
              text(10) TYPE c VALUE 'Text Field',
             END OF struc.
```

Correct syntax:

```
        DATA: BEGIN OF struc,
              text1(12)  TYPE c VALUE 'Text Literal',
              blanks(10) TYPE c VALUE IS INITIAL,
text2(10)  TYPE c VALUE 'Text Field',
              END OF struc.
```

Reason:

It must be possible to address all components of a structure explicitly. If you use literals or the special name SPACE in a structure definition, the system inserts nameless text fields as components. The intial value and length of the components are based on the contents and length of the literal. If you use SPACE, the system creates a text field filled with spaces. You cannot address these anonymous text fields explicitly in a program. In particular, a structure can never have a component SPACE. You can only address anonymous components using the structure name and offset and length. You can easily replace the anonymous components with named components. Named components can be addressed explicitly, but they can still have the same function as an anonymous field, namely to act as a "filler" field.

NON-LOCAL not allowed

The addition NON-LOCAL of the statements DATA, STATICS and CONSTANTS is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

METHOD FUNCTION FORM ...

```
DATA f TYPE ... NON-LOCAL.

 ...

ENDMETHOD ENDFUNCTION ENDFORM.
```

Correct syntax:

```
DATA f TYPE ...

METHOD FUNCTION FORM ...

...

ENDMETHOD ENDFUNCTION ENDFORM.
```

Reason:

The undocumented addition NON-LOCAL changes the attributes of a class or the local data objects of a procedure into global data objects of the main program. However, global data objects can only be declared in the main program. In particular, a class pool must not contain any global data objects; this could be circumvented by using the addition NON-LOCAL.

No definition of field groups in methods

The statement FIELD-GROUPS is not allowed in methods.

Error message in methods if the following syntax is used:

```
METHOD ...

...

FIELD-GROUPS fg.

...

ENDMETHOD.
```

Reason:

An extract dataset currently exists only as a global object of the main program. Therefore the field groups can only be defined globally in the main program. However, the definition of the field group structure, which is generated at runtime by the statement INSERT ... INTO fg, can also be executed in methods.

FIELDS not allowed

The statement FIELDS is not allowed.

Error message in ABAP Objects if the following syntax is used:

```
FIELDS f.
```

Correct syntax:

New pseudo comment for the extended program check.

Reason:

FIELDS no longer has any operational significance but is merely used as a note for the extended program check.

STATICS not allowed in instance methods

The statement STATICS is not allowed in instance methods.

Error message in ABAP Objects in the following case:

```
METHOD ...

STATICS s ...

...

ENDMETHOD.
```

Reason:

The statement STATICS in a method corresponds to a CLASS-DATA statement. However, visibility of the declared data objects is restricted to the method. With instance methods this may cause misunderstandings.

Assignments

CLEAR WITH NULL not allowed

The statement CLEAR WITH NULL is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
CLEAR f WITH NULL.
```

Correct syntax:

```
CONSTANTS hex(1) TYPE x VALUE IS INITIAL.

CLEAR f with hex.
```

Reason:

Initialization with an incompatible type must be avoided. If required, the statement CLEAR WITH NULL can be replaced by the above sequence of statements.

PACK not allowed

The statement PACK for assigning character fields is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
DATA: c1(...) TYPE c,
                p1(...) TYPE p.
 PACK c1 TO p1.
```

Correct syntax:

```
DATA: c1(...) TYPE c,
      p1(...) TYPE p.
MOVE c1 TO p1.
```

Reason:

The statement PACK works like the statement MOVE when a character field is assigned to a packed number and is thus obsolete.

MOVE PERCENTAGE not allowed

The statement MOVE PERCENTAGE is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
MOVE c1 TO c2 PERCENTAGE n.
```

Correct syntax:

```
DATA l TYPE i.
DESCRIBE FIELD c1 LENGTH l.
l = l * n / 100.
MOVE c1(l) TO c2.
```

Reason:

If required, assignment of a field percentage to another field can be implemented using other statements.

No arithmetic calculations using identical names of structure components

The statements for calculations with components that have the same name in two structures are not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
ADD-CORRESPONDING struc1 TO struc2.
DIVIDE-CORRESPONDING struc1 BY struc2.
MULTIPLY-CORRESPONDING struc1 BY struc2.
SUBTRACT-CORRESPONDING struc1 FROM struc2.
```

Reason:

The calculations are performed irrespective of the data type of the components. The identity of names cannot always ensure correct results or may cause runtime errors. The calculations should be programmed explicitly instead.

Comment:

The same actually applies to the statements MOVE-CORRESPONDING and the addition INTO CORRESPONDING FIELDS of the SELECT statement. However, for practical reasons it is not yet possible to forbid these statements. Nevertheless MOVE-CORRESPONDING should also be replaced by explicit assignments. The components of structures to be assigned can be combined in substructures both in the source and target structures; the substructures can then be assigned. If a structure cannot have substructures because it has been typed with reference to a database table, the same effect can be achieved using group names (addition AS NAME with the statement INCLUDE STRUCTURE TYPE).

The addition INTO CORRESPONDING FIELDS of the SELECT statement should be avoided at least in the static case (for performance reasons alone) and should be replaced by explicit field names in the INTO clause.

No summing-up of memory positions

The variants ADD THEN ... UNTIL ... and ADD FROM ... TO ... of the statement ADD are not allowed in ABAP Objects.

In ABAP Objects an error message is issued in the following cases:

```
ADD f1 THEN f2 UNTIL fn GIVING sum.
```

Correct syntax:

```
DO n TIMES VARYING f FROM f1 NEXT f2.

sum = sum + f.

ENDDO.
```

Reason:

The functionality of these statements depends on the internal structure of the program's working memory. Such operations should be avoided. If required, it is currently still possible to use the statements DO VARYING and WHILE VARY as a replacement.

No date conversions

The statements CONVERT DATE and CONVERT INVERTED DATE are not allowed.

Error message in ABAP Objects if the following syntax is used:

```
CONVERT DATE f1 INTO INVERTED-DATE f2.

CONVERT INVERTED-DATE f2 INTO DATE f1.
```

Correct syntax:

```
CONSTANTS comp_nine(20) TYPE c VALUE '09182736455463728190'.

f2 = f1.

TRANSLATE f2 USING comp_nine.

f1 = f2.

TRANSLATE f1 USING comp_nine.
```

Reason:

Date conversions are used mainly to influence the sort sequence in in internal tables. This function can be replaced by the additions ASCENDING or DESCENDING of the statement SORT. If required, you can easily program the nines complement yourself using TRANSLATE.

Offset/Length Specifications

No length specifications less than or equal to zero

For offset/length access to fixed-length fields, length specifications less than or equal to zero are not allowed.

Error message in ABAP Objects if the following syntax is used:

```
MOVE f+off(0) TO g.

MOVE f+5(-2) TO g.
```

Reason:

The length of a field section is always a positive amount. Fields with a fixed length of zero are not supported in ABAP. A fixed-length field always has a minimum length of one.

Note:

As of Release 4.6A the data types STRING and XSTRING for variable-length character or byte sequences are available. Empty character and byte sequences have a length of zero. Offset/Length accesses with zero length specifications have not been implemented yet for character and byte sequences.

Processing Character Strings

Only character-like fields in character string processing. To process character strings, only character-like fields (data types C, D, N, STRING, T and   in non-Unicode systems X and XSTRING) can be used.

Error message in ABAP Objects if the following syntax is used:

```
DATA int TYPE i.

SHIFT int BY 3 PLACES.
```

Correct syntax:

```
DATA int(4) TYPE c.
```

```
SHIFT int BY 3 PLACES.
```

Reason:

The statements for character string processing treat their operands as character-like irrespective of their actual type, which may lead to undefined results in connection with fields that are not character-like.

Field Symbols

No field symbols as class components. Field symbols must not be declared as class components in the declaration part of classes. Within methods, however, it is possible to create local field symbols.

Error message in the declaration part of classes if the following syntax is used:

```
FIELD-SYMBOLS ...
```

Correct syntax:

```
DATA ... TYPE REF TO ..

oder

ALIASES ...
```

Reason:

The only components allowed in classes are attributes, methods, and events. Field symbols are symbolic names for other fields. They use value semantics. In ABAP Objects, their role as a pointer is assumed by reference variables. Their role as a symbolic name can be replaced by alias names. Their role as a symbolic name can be replaced by alias names.

No field symbols without type assignment

In ABAP Objects the addition TYPE in the statement FIELD-SYMBOLS is mandatory. Error message in ABAP Objects if the following syntax is used:

```
FIELD-SYMBOLS <fs>.
```

Correct syntax:

```
FIELD-SYMBOLS <fs> TYPE ANY.
```

Reason:

Like method interface parameters, field symbols must always have explicit type assignments.

No obsolete casting with ASSIGN

In ABAP Objects it is not allowed to use the additions TYPE and DECIMALS for assigning data objects to field symbols using ASSIGN.

Error message in ABAP Objects if the following syntax is used:

```
ASSIGN f TO <fs> TYPE ... [DECIMALS ...].
```

Correct syntax:

```
ASSIGN f TO <fs> CASTING TYPE LIKE ...
```

Reason:

The TYPE and DECIMALS additions are replaced by the addition CASTING [TYPE LIKE]. Unlike the TYPE addition, CASTING supports any kind of data types.

No obsolete casting with FIELD SYMBOLS

In ABAP Objects the addition STRUCTURE is not allowed in the statement FIELD-SYMBOLS.

Error message in ABAP Objects if the following syntax is used:

```
FIELD-SYMBOLS <fs> STRUCTURE struc DEFAULT f.
```

Correct syntax:

```
FIELD-SYMBOLS <fs> TYPE LIKE struc.

ASSIGN f TO <fs> CASTING.
```

Reason:

Field symbols defined by the STRUCTURE addition are a mixture of field symbols with type assignment and a means for casting to local data types in a program or data types defined in the ABAP Dictionary. However, for type assignment to field symbols, the TYPE addition of the statement FIELD-SYMBOLS is available, and the addition CASTING of the statement ASSIGN can be used for casting.

No local copies using ASSIGN

In ABAP Objects it is not possible to use field symbols in procedures to work with copies of other data.

Error message in ABAP Objects if the following syntax is used:

```
ASSIGN LOCAL COPY OF INITIAL f TO <fs>.

ASSIGN LOCAL COPY OF f TO <fs>.
```

Correct syntax:

```
DATA dref TYPE REF TO data.

CREATE DATA dref LIKE f.

ASSIGN dref->* TO <fs>.

DATA dref TYPE REF TO data.

CREATE DATA dref LIKE f.

ASSIGN dref->* TO <fs>.

<fs> = f.
```

Reason:

Due to the introduction of the general data references concept, the addition LOCAL COPY of the statement ASSIGN is obsolete. The value of f can be copied using the assignment <fs> = f after ASSIGN.

No search limitations with dynamic ASSIGN

The search limited to table work areas of the main program in connection with the dynamic ASSIGN statement is no longer used in ABAP Objects.

Error message in ABAP Objects in the following case:

```
ASSIGN TABLE FIELD (<f>) TO <fs>.
```

Correct syntax:

```
ASSIGN (<f>) TO <fs>.
```

Cause:

Table work areas are not supported in classes.

Logical Expressions and Control Structures

Incorrect logical operators

The logical operators ><, =< and => are not allowed in ABAP Objects. This also applies to logical expressions in the addition WHERE of the LOOP statement for internal tables and in the WHERE clause of Open-SQL statements.

Error message in ABAP Objects if the following syntax is used:

```
... >< ... =< ... => ...
```

Correct syntax:

```
... <> ... <= ... >= ...
```

Reason:

These operators for NOT EQUAL, LESS/EQUAL and GREATER/EQUAL are obsolete. They have the same function as <>, <= and >= (or NE, LE and GE).

ON CHANGE OF - ENDON not allowed

The pseudo control structure ON CHANGE OF - ENDON is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
ON CHANGE OF f.

...

ENDON.
```

Correct syntax:

```
DATA g LIKE f.

IF f <> g.

...

g = f.

ENDIF.
```

Reason:

A global invisible work field over which the program has no control is created internally. A separate work field should be declared and processed using the IF control structure.

Incorrect statement after CASE

In ABAP Objects WHEN must be the first statement after CASE.

Error message in ABAP Objects if the following syntax is used:

```
CASE a.

MOVE 5 TO a.

WHEN 5.

WRITE a.

ENDCASE.
```

Correct syntax:

```
MOVE 5 TO a.

CASE a.

WHEN 5.

WRITE a.

ENDCASE.
```

Reason:

The CASE control structure must always reflect the semantics of a IF-ELSEIF control structure, which is not ensured if a statement may come between CASE and WHEN.

Internal Tables

Declaration with OCCURS not allowed

In ABAP Objects it is not allowed to use the addition OCCURS in the statements TYPES and DATA (and the other declarative statements) to define internal tables. Error message in ABAP Objects if the following syntax is used:

```
TYPES DATA: BEGIN OF itab OCCURS n,

      ...

      fi ...,

      ...

      END OF itab.

      and

      TYPES DATA itab TYPE LIKE line_type OCCURS n.
```

Correct syntax:

```
TYPES DATA: BEGIN OF line_type,

      ...

      fi ...,

      ...

      ...

      END OF line_type.


      TYPES itab TYPE LIKE STANDARD TABLE OF line_type

            WITH NON-UNIQUE DEFAULT KEY

            [INITIAL SIZE n].


     DATA itab TYPE LIKE [STANDARD] TABLE OF line_type

               [INITIAL SIZE n].
```

Reason:

The new addition TYPE LIKE TABLE OF of the statements DATA and TYPES makes the addition OCCURS for the table declaration obsolete. If required, initial memory requirements can be specified using the addition INITIAL SIZE.

Note:

The short form DATA itab TYPE LIKE TABLE of line_type. is supplemented by the system as follows:

```
DATA itab TYPE LIKE STANDARD TABLE OF line_type
         WITH NON-UNIQUE DEFAULT KEY
         INITIAL SIZE 0.
```

It can be used instead.

The short form

```
TYPES itab TYPE LIKE STANDARD TABLE of line_type.
```

is supplemented by the system as follows:

```
TYPES itab TYPE LIKE STANDARD TABLE of line_type
         INITIAL SIZE 0.
```

It defines a standard table type with a generic key that can be used to assign types to interface parameters and field symbols.

Tables with header lines not allowed

Only tables without header lines can be declared in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
DATA itab TYPE LIKE TABLE OF ... WITH HEADER LINE.
```

Correct syntax:

```
DATA: itab TYPE LIKE TABLE OF ... ,
wa LIKE LINE OF itab.
```

Reason:

It depends on the statement whether the body or header line of a table is accessed. The table name should identify the table uniquely. Programs are easier to read. Tables with header lines do not improve performance.

Note:

When calling external procedures (subroutines and function modules), which contain TABLES parameters in their parameter interface, please note that a TABLES parameter always requires a header line in addition to a table body. When a table without header line is transferred, the header line of the TABLES parameter retains its initial value. When calling such procedures in methods, you have to check whether transfer of the header line is expected in the procedure and evaluated there. You may have to adjust or rewrite the procedure. Method interfaces do not have TABLES parameters.

Short forms not allowed with line operations in Internal Tables

Short forms for operations on table lines are not allowed in ABAP Objects. An explicit work area or field symbol must always be used.

Error message in ABAP Objects if the following syntax is used:

Operations for all table types:

```
INSERT TABLE itab.
COLLECT itab.
READ TABLE itab ...
MODIFY TABLE itab ...
MODIFY itab ... WHERE ...
DELETE TABLE itab.
LOOP AT itab ...
```

Operations for index tables

```
APPEND itab.
INSERT itab ...
MODIFY itab ...
```

Correct syntax:

Operations for all table types:

```
INSERT wa INTO TABLE itab.
COLLECT wa INTO itab.
READ TABLE itab ... INTO wa   ASSIGNING <fs>.
MODIFY TABLE itab FROM wa ...
MODIFY itab FROM wa ... WHERE ...
DELETE TABLE itab FROM wa.
LOOP AT itab INTO wa ...   ASSIGNING <fs> ...
```

Operations for index tables:

```
APPEND wa TO itab.
INSERT wa INTO itab ...
MODIFY itab FROM wa ...
```

Reason:

Clear separation of table and work area. Programs are easier to read. Since only tables without header lines can be declared in classes, this is a limitation only when global tables of the main program are accessed in local classes.

No changes of internal tables in loops

In ABAP Objects it is not allowed to change a complete internal table within a loop on the same table. Changes of complete tables are initiated, for example, by the statements REFRESH, CLEAR, FREE, MOVE, SORT or SELECT INTO TABLE. This also applies to transferring internal tables from procedures or importing internal tables from data clusters.

Error message in ABAP Objects if the following syntax is used:

```
LOOP AT itab INTO wa.

CLEAR itab.

ENDLOOP.
```

Correct syntax:

```
LOOP AT itab INTO wa.

...

ENDLOOP.

CLEAR itab.
```

Reason:

When tables are accessed again, the reaction is undefined and may cause runtime errors.

Compatible line types with INSERT INTO TABLE

For inserting lines of an internal table into another internal table, the line types must be compatible in ABAP Objects.

Error message in ABAP if the following syntax is used:

```
DATA: itab TYPE SORTED TABLE OF f

        WITH UNIQUE KEY table_line,

jtab TYPE HASHED TABLE OF i

        WITH UNIQUE KEY table_line.



INSERT LINES OF itab INTO TABLE jtab.
```

Correct syntax:

```
DATA: itab TYPE SORTED TABLE OF f

        WITH UNIQUE KEY table_line,

jtab TYPE HASHED TABLE OF f

        WITH UNIQUE KEY table_line.
```

INSERT LINES OF itab INTO TABLE jtab.

Reason:

For all generic insert operations (inserting lines into all table types) the lines to be inserted must be compatible with the line type of the table. The above statement is adjusted accordingly.

Compatible work area with control level processing

For control level processing of an internal table in ABAP Objects, the work area must be compatible with the line type of the table.

Error message in ABAP Objects if the following syntax is used:

```
DATA: itab LIKE TABLE OF line,

        wa(255) TYPE x.

        SORT itab by col1.

        LOOP AT itab INTO wa.

        AT NEW col1.

        ENDAT.

        ENDLOOP.
```

Correct syntax:

```
DATA: itab LIKE TABLE OF line,

        wa   LIKE LINE OF itab.

        SORT itab by col1.

        LOOP AT itab INTO wa.

        AT NEW col1.

        ENDAT.

        ENDLOOP.
```

Reason:

Control level processing is based on the line structure of the internal table. To determine the control break, the system evaluates the work area, which must therefore have the same structure as a table line.

No obsolete work area

In ABAP Objects it is not allowed to specify a work area with READ TABLE if the addition TRANSPORTING NO FIELDS is used.

Error message in ABAP Objects if the following syntax is used:

```
READ TABLE itab INDEX i INTO wa TRANSPORTING NO FIELDS.
```

Correct syntax:

```
READ TABLE itab INDEX i TRANSPORTING NO FIELDS.
```

Reason:

The specification INTO wa is obsolete. It does not affect the work area.

No obsolete key specification with TABLE LINE

If the complete line of an internal table is to be specified as a key, it is not allowed to specify TABLE LINE in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
... TABLE LINE ...
```

Correct syntax:

```
... table_line ...
```

Reason:

The pseudo component table_line replaces the construct TABLE LINE.

Obsolete READ Variants not allowed

The READ variants in which key values are read from header lines are not allowed in ABAP Objects.

In ABAP Objects, an error message occurs on:

```
READ TABLE itab.
```

Correct syntax:

```
READ TABLE itab FROM key INTO wa.
```

Or

```
READ TABLE itab WITH KEY ... INTO wa.
```

Reason:

These variants use an implicit key that consists of all fields of the header line of the table that are neither numeric (type I, P, or F), nor tables, and that do not contain the value SPACE. Instead, y you should specify the key explicitly. The variant only applied to internal tables with header line.

The READ variants, WITH KEY is obsolete in ABAP Objects.

In ABAP Objects, an error message occurs on:

```
READ TABLE itab WITH KEY key INTO wa.
```

Reason:

The key fields of a table should always be components of the line structure.

The READ variant, in which the entire table line is addressed as a component and where the specified key value is compared with the entire table line is not allowed in ABAP Objects.

In ABAP Objects, an error message occurs on:

```
READ TABLE itab WITH KEY = key INTO wa.
```

Correct syntax:

```
READ TABLE itab WITH KEY table_line = key INTO wa.
```

Reason:

This variant is a special solution to allow key access to tables without a structured line type. The introduction of the pseudocomponent table_line, which can always be used instead of a key field, makes this variant of the READ statement redundant. When you use an explicit search key in the READ TABLE statement, you can only specify a table column once.

In ABAP Objects, an error occurs on:

```
READ TABLE itab INTO line WITH KEY col1 = ... col1 = ...
```

Correct syntax:

```
READ TABLE itab INTO line WITH KEY col1 = ...
```

Reason:

Only the last specification is used. Multiple occurrences are redundant.

No WRITE TO for internal tables

The statement WRITE TO is not allowed for internal tables in ABAP Objects.

Error message from ABAP Objects in the following case:

```
WRITE ... TO itab INDEX idx.
```

Correct syntax:

```
FIELD-SYMBOLS <fs> TYPE ...

READ TABLE itab INDEX idx ASSIGNING <fs>.

WRITE ... TO <fs>.
```

Reason:

Field symbols can be used for direct access to table lines. The statement WRITE TO for table lines is obsolete.

No field symbols as sort criteria

In ABAP Objects it is not allowed to use field symbols as sort criteria for sorting internal tables.

Error message in ABAP Objects if the following syntax is used:

```
name = 'ITAB-COL1'.

ASSIGN (name) TO <fs>.

SORT itab BY <fs>.
```

Correct syntax:

```
name = 'COL1'.

SORT itab BY (name).
```

Reason:

Sort criteria must be specified with reference to the line structure (columns) of an internal table. Field symbols point to data objects and must not be used for naming structure components. Since dynamic name specification is possible, it is not necessary to specify columns using field symbols that point to the work area. This variant was only available anyway for tables with a header line.

No field symbols as a control break criterion

In ABAP Objects it is not allowed to specify field symbols as control break criteria in control level processing.

Error message in ABAP Objects if the following syntax is used:

```
name = 'WA-COL1'.

ASSIGN (name) TO <fs>.

LOOP AT itab INTO wa.

AT NEW <fs>.

...

ENDAT.

ENDLOOP.
```

Correct syntax:

```
name = 'COL1'.

LOOP AT itab INTO wa.

AT NEW (name).

...

ENDAT.

ENDLOOP.
```

Reason:

Control break criteria must be specified with reference to the line structure (columns) of the internal table. Field symbols point to data objects and must not be used for naming structure components. Since dynamic name specification is possible, column specifications via field symbols that point to the used work area are obsolete.

INFOTYPES not allowed

This statement creates a specific table with a header line and is therefore not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
INFOTYPES nnn.
```

Reason:

Since tables with header lines are not allowed at all in ABAP Objects, the required table must be defined using permissible statements.

RANGES not allowed

This statement creates a specific table with a header line and is therefore not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
RANGES rangetab FOR f.
```

Correct syntax:

```
DATA rangetab TYPE LIKE RANGE OF ...
```

Reason:

Since tables with header lines are not allowed at all in ABAP Objects, the required table must be declared using permissible statements.

PROVIDE - ENDPROVIDE not allowed

These statements can only be used for tables with header lines and are therefore not allowed in ABAP Objects.

Error message in ABAP if the following syntax is used:

```
PROVIDE f1 f2 ... FROM itab1

g1 g2 ... FROM itab2

...

FROM itabn

...

BETWEEN f AND g.

...

ENDPROVIDE.
```

Reason:

The statements can only be used in local classes for tables in the main program. However, working with global data is not desired in an ABAP Objects context. Until a solution is available which is compatible with object-oriented semantics, customers have to program the functionality of PROVIDE - ENDPROVIDE, or the statements must be transferred to external procedures.

Procedures

No interface parameters without type assignment

In ABAP Objects the addition TYPE is mandatory for interface parameters of procedures. The only procedures allowed in ABAP Objects are methods.

Error message in ABAP Objects if the following syntax is used:

```
METHODS meth IMPORTING p1

        EXPORTING p2.
```

Correct syntax:

```
METHODS meth IMPORTING p1 TYPE ANY

        EXPORTING p2 TYPE ANY.
```

Reason:

In ABAP Objects the interface parameters of procedures must always have explicit type assignments. If you require an interface parameter which is completely generic, you can use TYPE ANY.

Reference transfer is standard for methods

In contrast to function modules, reference transfer is the standard for method interface parameters. To pass a parameter as a value, only the name p must be specified in the interface definition instead of VALUE(p). For the return value (RETURNING parameter) only the value can be passed on.

Reason:

Performance advantages of reference transfer as compared to value transfer.

Static check of interface parameters

In contrast to function modules, the specified interface parameters are already checked before the static syntax check when a method is called. This is true both for local and global classes. The system checks whether only valid formal parameters have been specified, whether the actual parameters have the correct type and whether all mandatory input parameters are supplied.

Reason:

Reduction of runtime errors during program execution. The interface check within the extended program check is not always sufficient.

Exceptions must be defined in the triggering class

In contrast to function modules, only those exceptions can be triggered in a method that have been defined for this method. The static syntax check checks whether the exception exists (when an exception is triggered by the statement RAISE and when it is handled using the exception EXCEPTIONS of the statement CALL METHOD). A non-existent exception raised in a function module merely causes a syntax warning message, and handling of a non-existent exception in the statement CALL FUNCTION is not checked at all. Therefore undefined exceptions can be raised and handled in function modules.

Reason:

Runtime errors caused by triggering non-existent exceptions are prevented.

No definition of interface parameters with TABLES

In ABAP Objects the construct TABLES for defining formal parameters in procedures is not allowed. The only procedures allowed in ABAP Objects are methods.

Error message in ABAP Objects in the following case:

```
METHODS meth TABLES p1.
```

Correct syntax:

```
METHODS meth CHANGING p1 LIKE itab.
```

Reason:

Like all data objects, internal tables are transferred as IMPORTING, EXPORTING, CHANGING or RETURNING parameters.

Old casting not allowed for Interface Parameters

In ABAP Objects, the STRUCTURE addition for specifying the types of formal parameters is not allowed. The only procedures permitted in ABAP Objects are methods.

In ABAP Objects, an error message occurs on:

```
METHODS meth IMPORTING p1 STRUCTURE struc.
```

Correct syntax:

```
METHODS meth IMPORTING p1 TYPE struc.
```

Or

```
METHODS meth IMPORTING p1 TYPE ANY.

METHOD meth.

FIELD-SYMBOLS <fs> TYPE struc.

ASSIGN p1 TO <fs> CASTING.

...

ENDMETHOD.
```

Reason:

Formal parameters typed using STRUCTURE are a mixture of typed parameters and parameters with casting to local or ABAP Dictionary types. However, you should use the TYPE addition to specify the types of field symbols. For casting, use local field symbols and the ASSIGN ... CASTING statement.

Statement LOCAL not allowed

The statement LOCAL is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
LOCAL f.
```

Reason:

The statement LOCAL can be used only in subroutines anyway. It protects global data objects against changes in the subroutine. This is not appropriate for methods because methods are intended to work with the class attributes. For internal purposes, local data objects can be created both in methods and subroutines.

PERFORM form(prog) not allowed

In ABAP Objects it is not allowed to specify external subroutines using the syntax form(prog).

Error message in ABAP Objects if the following syntax is used:

```
PERFORM form(prog) ...
```

Correct syntax:

```
PERFORM form IN PROGRAM prog ...
```

Reason:

The statement PERFORM form(prog) is replaced by the statement PERFORM form IN PROGRAM prog. Unlike form(prog), the name specification form IN PROGRAM prog supports the specification of dynamic program names using form IN PROGRAM (name). The static form form(prog) does not correspond to the usual ABAP semantics where the dynamic form is distinguished from the static form through parentheses.

Passing SY-REPID not allowed

In ABAP Objects, you cannot pass the system field SY-REPID as an actual parameter to an external procedure.

In ABAP Objects, an error message occurs on:

```
CALL FUNCTION func EXPORTING p = sy-repid.
```

Correct syntax:

```
DATA repname TYPE sy-repid.

repname = sy-repid.

CALL FUNCTION func EXPORTING p = repname.
```

Reason:

When the parameters are passed to the formal parameters, SY-REPID already contains the name of the main program of the procedure you have called, even though you intended to pass the name of the calling program.

No transfer of SY-SUBRC

The system field SY-SUBRC must not be used in ABAP Objects as an actual parameter for the output parameters of external procedures.

Error message in ABAP Objects in the following case:

```
CALL FUNCTION func IMPORTING p = sy-subrc.
```

Correct syntax:

```
DATA subrc TYPE sy-subrc.

CALL FUNCTION func IMPORTING p = subrc.
```

Reason:

After parameter transfer, SY-SUBRC is set by the call statement. This overwrites the transferred value. With a few exceptions, system fields should never be overwritten explicitly in a program.

Data Clusters

Identification must be specified

The addition ID must be specified with the statements

```
IMPORT/EXPORT/FREE ... MEMORY in ABAP Objects.
```

Error message in ABAP Objects if the following syntax is used:

```
EXPORT f TO MEMORY.

IMPORT f FROM MEMORY.

FREE MEMORY.

Correct syntax:

EXPORT f TO MEMORY ID key.

IMPORT f FROM MEMORY ID key.

FREE MEMORY ID key.
```

Reason:

If no identification is specified, all programs in a call chain use the same memory area. This may lead to unpredictable results especially with complex transactions.

No generic identification

The additions MAJOR-ID and MINOR-ID for reading cluster databases are not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
IMPORT ... FROM DATABASE dbtab(ar) ... MAJOR-ID maid

[MINOR-id miid].
```

Correct syntax:

```
IMPORT ... FROM DATABASE dbtab(ar) ID id.
```

Reason:

The data cluster identification specified in the statement should be unique. If required, the logic in the additions MAJOR-ID and MINOR-ID must be programmed by the user

No implicit field names in cluster

When data clusters are exported and imported, explicit names must be specified in ABAP objects for the exported data objects. The notation with an equals sign (=) should be preferred to the old additions FROM and TO.

Error message in ABAP Objects if the following syntax is used:

```
EXPORT f1 ... fn TO ...

IMPORT f1 ... fn FROM ...
```

Correct syntax:

```
EXPORT name1  =   f1 ... namen   =  fn TO ...

IMPORT name1  =   f1 ... namen   =  fn FROM ...
```

Or

```
EXPORT name1 FROM f1 ... namen FROM fn TO ...

IMPORT name1 TO   f1 ... namen TO   fn FROM ...
```

Reason:

The use of implicit names may lead to errors. The specified names are to be understood as field identifications in the cluster. With the implicit method, the specified identifications are used literally when data clusters are exported, i.e., including offset/length specifications or preceding selectors. When clusters are imported in a different context, these identifications must be known and must be specified identically. Because of the similarity to method and function module calls, the use of the equals sign (=) emphasizes that formal parameters appear on the left and actual parameters on the right. The expressions FROM and TO are also additionally used to specify where data is stored.

No use of table work areas

If user data fields are to be transported to or from cluster databases or the cross-transaction application buffer, the addition FROM/TO wa must be specified with the statements EXPORT/IMPORT ... TO/FROM DATABASE/SHARED BUFFER in ABAP Objects.

The following cannot be used in ABAP Objects:

```
dbtab-... = ...

EXPORT ... TO DATABASE dbtab(ar) ID id.

EXPORT ... TO SHARED BUFFER dbtab(ar) ID id.

IMPORT ... FROM DATABASE dbtab(ar) ID id.

IMPORT ... FROM SHARED BUFFER dbtab(ar) ID id.

... = dbtab-
```

Correct syntax:

```
DATA wa TYPE dbtab.

WA-... = ...

EXPORT ... TO DATABASE dbtab(ar) ID id FROM wa.

EXPORT ... TO SHARED BUFFER dbtab(ar) ID id FROM wa.
```

```
IMPORT ... FROM DATABASE dbtab(ar) ID id TO wa.

IMPORT ... FROM SHARED BUFFER dbtab(ar) ID id TO wa.

... = wa-...
```

Reason:

If the addition FROM/TO wa is not specified, the system tries to transport the user data fields of the table work area declared by TABLES. If the table work area is not found, the user data fields are not transported. A table work area can be declared in the main program of local classes; however, it should not be used. In global classes the table work area cannot be used at all.

No clusters in files

Exporting and importing of file clusters to and from files on the application server has not been released yet in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
EXPORT ... TO DATASET ...

IMPORT ... FROM DATASET ...
```

Reason:

These statements have not yet been implemented fully and cannot be used until a later release.

Program Calls

Incorrect transaction call

The additions USING and AND SKIP FIRST SCREEN for the statement CALL TRANSACTION are mutually exclusive.

Error message in ABAP Objects if the following syntax is used:

```
CALL TRANSACTION ... USING itab AND SKIP FIRST SCREEN.
```

Correct syntax:

```
CALL TRANSACTION ... USING itab.
```

Reason:

The content of the batch input table specified in the addition USING controls the entire transaction flow including the display of screens. The addition AND SKIP FIRST SCREEN is to be used only in connection with filling the mandatory input fields via SPA/GPA parameters.

No implicit field names when calling dialog modules

To call dialog modules, the names of the fields from or to which data is transferred must be specified explicitly in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
CALL DIALOG ... EXPORTING f1 ... fn

IMPORTING f1 ... fn.
```

Correct syntax:

```
CALL DIALOG ... EXPORTING f1 FROM f1 ... fn FROM fn

IMPORTING f1 TO   f1 ... fn TO   fn.
```

Reason:

The use of implicit names may cause errors. With the implicit method, the system tries to find global data objects in the called program whose names are literally the same as the names specified after FROM or TO. If offset/length specifications or preceding selectors are used in the names, data objects with identical names cannot exist in the called program.

Database Accesses

Short forms not allowed

For each SQL statement, explicit work areas must be specified in ABAP Objects. Data objects with an appropriate type can be created with reference to the database definition in the ABAP Dictionary.

Error message in ABAP Objects if the following syntax is used.

```
SELECT ... FROM dbtab

INSERT dbtab.

UPDATE dbtab.

DELETE dbtab.

ODIFY dbtab.
```

Correct syntax:

```
DATA wa TYPE dbtab.

SELECT ... FROM dbtab INTO wa.

INSERT dbtab FROM wa.
```

Or

```
INSERT INTO dbtab VALUES wa.

UPDATE dbtab FROM wa.
```

Or

```
UPDATE dbtab SET ... .

DELETE dbtab FROM wa.
```

Or

```
DELETE FROM dbtab WHERE ...

MODIFY dbtab FROM wa.
```

Reason:

Clear separation of database name and ABAP work area. Programs are easier to read. To work with short forms, table work areas must be declared using the TABLES statement, which is not allowed in ABAP Objects.

Note:

This does not apply to SELECT statements in subqueries. For subqueries INTO clauses are not allowed. For more information, refer to the EXISTS construct of the WHERE and HAVING clauses of SELECT, UPDATE, DELETE and OPEN CURSOR.

Work area * not allowed

In ABAP Objects it is not allowed to specify * work areas as names for database tables and work areas.

Error message in ABAP Objects if the following syntax is used:

```
SELECT ... FROM *dbtab INTO ...

INSERT *dbtab.

UPDATE *dbtab.

DELETE *dbtab.

MODIFY *dbtab.
```

Correct syntax:

```
DATA wa TYPE dbtab.

SELECT ... FROM dbtab INTO wa.

INSERT dbtab FROM wa.
```

Or

```
INSERT INTO dbtab VALUES wa.

UPDATE dbtab FROM wa.
```

Or

```
UPDATE dbtab SET ... .

DELETE dbtab FROM wa.
```

Or

```
DELETE FROM dbtab WHERE ...

MODIFY dbtab FROM wa.
```

Reason:

The declaration of work areas that are suitable for the respective types using the DATA statement replaces the declaration of * work areas. * work areas can only be declared using the TABLES statement, which is not allowed in ABAP Objects. * work areas can only be used in the forbidden short forms of the Open-SQL statements.

READ TABLE not allowed

The statement READ TABLE for reading data from database tables is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
t100 = space.

t100-sprsl = 'D'.

t100-arbgb = 'BC'.

t100-msgnr = '100'.

READ TABLE t100.
```

Correct syntax:

```
DATA wa TYPE t100.

SELECT SINGLE * FROM t100 INTO wa WHERE sprsl = 'D'  AND

arbgb = 'BC' AND msgnr = '100'.
```

Reason:

The statement is replaced by the Open-SQL statement SELECT. It works only with database tables whose names correspond to the naming conventions for R/2-ATAB tables (five characters max. and T as the first character) and with table work areas declared by TABLES, which are not allowed in ABAP Objects. Generic key values are used for access, which are taken flush left from the used part of the table work area. Instead the key should be specified explicitly in the WHERE clause of the SELECT statement.

LOOP AT not allowed

The statement LOOP AT for reading data from database tables is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
t100 = space.

t100-sprsl = 'D'.

t100-arbgb = 'BC'.
```

```
t100-msgnr = '1'.

LOOP AT t100.

...

ENDLOOP.
```

Correct syntax:

```
DATA wa TYPE t100.

SELECT * FROM t100 INTO wa WHERE sprsl = 'D'  AND

arbgb = 'BC' AND

msgnr LIKE '1%'.

...

ENDSELECT.
```

Reason:

The statement is replaced by the Open-SQL statement SELECT. It works only with database tables whose names correspond to the naming conventions for R/2-ATAB tables (five characters max. and the character T in the first position) and with table work areas declared by TABLES, which are not allowed in ABAP Objects. Generic keywords are used for access, which are taken flush left from the used part of the table work area. Instead the key should be specified explicitly in the WHERE clause of the SELECT statement.

REFRESH FROM not allowed

The statement REFRESH FROM for reading data from database tables is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
t100 = space.

t100-sprsl = 'D'.

t100-arbgb = 'BC'.

t100-msgnr = '1'.

REFRESH itab FROM TABLE t100.
```

Correct syntax:

```
DATA wa TYPE t100.

SELECT * FROM t100 INTO TABLE itab WHERE sprsl = 'D'  AND

arbgb = 'BC' AND

msgnr LIKE '1%'.
```

Reason:

The statement is replaced by the Open-SQL statement SELECT. It works only with database tables whose names correspond to the naming conventions for R/2-ATAB tables (five characters max. and T as the first character) and with table work areas declared by TABLES, which are not allowed in ABAP Objects. Generic key values are used for access, which are taken flush left from the used part of the table work area. Instead, the key has to be specified explicitly in the WHERE clause of the SELECT statement.

VERSION addition not allowed

The VERSION addition in the OPEN SQL statements DELETE and MODIFY (and in the obsolete statements READ TABLE und LOOP AT) is forbidden in ABAP Objects.

In ABAP Objects, an error message will occur if you use:

```
DELETE dbtab VERSION vers.

MODIFY dbtab VERSION vers.

CONCATENATE 'T' vers INTO vers.

DELETE (vers) FROM dbtab.

MODIFY (vers) FROM dbtab.
```

Reason:

The VERSION addition only works with database tables whose names comply with the naming conventions for R/2 tables. VERSION is replaced by specifying table names dynamically using field names in parentheses.

Incorrect logical operators in the WHERE clause

The logical operators >< , =< and => are not allowed in ABAP Objects.

Error message in ABAP Objects in the following case:

```
... >< ... =< ... => ...
```

Correct syntax:

```
... <> ... <= ... >= ...
```

Reason:

These operators for 'not equal', 'less or equal' and 'greater or equal' are obsolete. They have the same functionality as

<> , <= and >= (or NE, LE and GE).

Subroutine Calls Not Allowed in EXEC SQL

Using the PERFORMING addition in the EXEC SQL statement to use a subroutine to process data line by line that you have read using Native SQL is not allowed in ABAP Objects. The EXIT FROM SQL statement, previously used within such subroutines, is also forbidden.

In ABAP Objects, an error message occurs on:

```
EXEC SQL PERFORMING form.
select ... into :wa from dbtab where ...
ENDEXEC.
FORM form.
...
EXIT FROM SQL.
...
ENDFORM.
```

Correct syntax:

```
EXEC SQL.
open c1 for
select ... from dbtab where ...
ENDEXEC.
DO.
EXEC SQL.
fetch next c1 into :wa
ENDEXEC.
IF sy-subrc <> 0.
EXIT.
ENDIF.
...
ENDDO.
EXEC SQL.
close c1
ENDEXEC.
```

Reason:

You should not call subroutines from local classes, and cannot call them from global classes. The called subroutine has no interface, working instead with the global data of the main program. The EXIT FROM SQL statement ends the SQL processing without reference to the actual SQL statement.

Lists

Incorrect dynamic position specifications

Dynamic position specifications with WRITE or ULINE in ABAP Objects are only possible following the addition AT.

Error message from ABAP Objects in the following case:

```
DATA: off TYPE i, len TYPE i.

WRITE /off(len) f.
```

Correct syntax:

```
DATA: off TYPE i, len TYPE i.

WRITE AT /off(len) f.
```

Reason:

Standardization of the syntax for the offset/length specifications in output statements. Since dynamic offset/length specifications without the line feed character (/) are allowed only after AT anyway, AT must always be used.

No obsolete formatting statements

Format specifications, for which the FORMAT statement could be used, are not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
        DETAIL.

        SUMMARY.

        INPUT.
```

Correct syntax:

```
FORMAT INTENSIFIED OFF.

FORMAT INTENSIFIED ON.

FORMAT INPUT ON.
```

Reason:

The statements are replaced by the additions of the FORMAT or WRITE statement and are therefore obsolete.

Automatic Calculations in WRITE not allowed

The MAXIMUM, MINIMUM, and SUMMING statements for calculating values automatically during creation of a basic list with the WRITE statement are not allowed in ABAP Objects.

In ABAP Objects, an error message occurs on:

```
MAXIMUM f.
MINIMUM f.
SUMMING f.
...
WRITE f.
...
WRITE: / max_f, min_f, sum_f.
```

Correct syntax:

```
DATA: max_f like f,
min_f like f,
sum_f like f.
...
WRITE f.
IF max_f < f.
max_f = f.
ENDIF.
IF min_f > f.
min_f = f.
ENDIF.
sum_f = sum_f + f.
...
WRITE: / max_f, min_f, sum_f.
```

Reason:

These statements create internal global variables max_f, min_f, and sum_f. To improve the readability of programs, they should no longer be used. Instead, you should program these functions yourself.

MARK not allowed

The statement MARK is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
MARK.
```

Reason:

The statement MARK comes from R2 and is undocumented. The documented statements for interactive list processing should be used instead. Print parameters missing with NEW-PAGE

In ABAP Objects the statement NEW-PAGE PRINT ON can only be used with print parameters specified unless a user dialog takes place.

Error message in ABAP Objects if the following syntax is used:

```
NEW-PAGE PRINT ON NO DIALOG.
```

Correct syntax:

```
DATA pripar TYPE pri_params,

arcpar TYPE arc_params.

CALL FUNCTION 'GET_PRINT_PARAMETERS'

IMPORTING out_parameters = pripar

out_archive_parameters = arcpar

...

NEW-PAGE PRINT ON NO DIALOG

PARAMETERS pripar

ARCHIVE PARAMETERS arcpar.
```

Reason:

Printing without dialog and without consistent print parameters causes incorrect print output. Missing print parameters with SUBMIT.

In ABAP Objects the statement SUBMIT ... TO SAP-SPOOL can only be executed with print paramameters specified unless a user dialog takes place.

Error message in ABAP Objects if the following syntax is used:

```
SUBMIT report TO SAP-SPOOL WITHOUT SPOOL DYNPRO.
```

Correct syntax:

```
DATA pripar TYPE pri_params,
arcpar TYPE arc_params.
CALL FUNCTION 'GET_PRINT_PARAMETERS'
IMPORTING out_parameters = pripar
out_archive_parameters = arcpar
...
SUBMIT report TO SAP-SPOOL WITHOUT SPOOL DYNPRO
SPOOL PARAMETERS pripar
ARCHIVE PARAMETERS arcpar.
```

Reason:

Printing without dialog and without consistent print parameters causes incorrect print output.

NEW-SECTION not allowed

The statement NEW-SECTION is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
NEW-SECTION.
```

Correct syntax:

```
NEW-PAGE PRINT ON NEW-SECTION.
```

Reason:

Reset of the line counter can be controlled using the statement NEW-PAGE PRINT ON. The statement NEW-SECTION is obsolete.

No constants in the HIDE area

Only variables can be written to the HIDE area in ABAP Objects.

Reason:

Printing without dialog and without consistent print parameters causes incorrect print output.

NEW-SECTION not allowed

The statement NEW-SECTION is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
NEW-SECTION.
```

Correct syntax:

```
NEW-PAGE PRINT ON NEW-SECTION.
```

Reason:

Reset of the line counter can be controlled using the statement NEW-PAGE PRINT ON. The statement NEW-SECTION is obsolete.

No constants in the HIDE area

Only variables can be written to the HIDE area in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
CONSTANTS f.

HIDE: '...', f.
```

Correct syntax:

```
DATA: f1, f2.

HIDE: f1, f2.
```

Reason:

In connection with interactive list events, the fields hidden by HIDE are overwritten by the values from the HIDE area and must therefore be changeable.

Reporting

STOP not allowed

The statement STOP is not allowed in ABAP Objects.

Error message in ABAP Objects in the following case:

```
STOP.
```

Reason:

The statement STOP is intended for the events INITIALIZATION, AT SELECTION-SCREEN, START-OF-SELECTION and GET. During the execution of type-1 programs, it stops processing of the associated event blocks and triggers the runtime environment event END-OF-SELECTION. The event concept of the ABAP runtime environment is not supported by ABAP Objects.

REJECT not allowed

The statement REJECT is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

```
REJECT [dbtab].
```

Reason:

The statement REJECT is intended only for use in connection with the event GET during execution of type-1 programs with logical databases. REJECT stops processing of the current line of the node in the logical database and reads the next line of the same node or lower node dbtab. The previous event concept of the ABAP runtime environment and thus the previous handling of logical databases is not supported by ABAP Objects.

CHECK SELECT-OPTIONS not allowed

The construct SELECT-OPTIONS in the statement CHECK is not allowed in ABAP Objects.

Error message in ABAP Objects if the following syntax is used:

CHECK SELECT-OPTIONS.

Correct syntax:

CHECK f IN seltab.

Reason:

This form of the statement CHECK is intended only for use during the event GET during execution of type-1-programs with logical databases. The statement checks whether the content of the work area, which was filled by the logical database for the current GET event, meets the conditions in all the selection tables that are connected with the database table read. The name of the database table is taken statically from the next higher GET statement in the ABAP program. Thus the statement does not make sense outside of an GET event block. However, the previous event concept of the ABAP runtime environment, that is, the previous way of processing logical databases, is not supported by ABAP Objects.

Author Bio

When not actively engaged as a grandmother and mother, Marilyn Pratt serves as a Community Manager of the SAP Developer Network and can be found lurking in the ABAP forum, XI pages, and other Technology areas.