boolc, boolx, xsdbool - Boolean Functions

Variants:

1. ... boolc(log_exp) ...

2. ... boolx(bool = log_exp bit = bit) ...

3. ... xsdbool(log_exp) ...

Effect

The Boolean functions determine the truth value of a logical expression **log_exp** specified as an argument. For **log_exp**, any logical expression can be specified in accordance with the applicable rules. The return value of a Boolean function has a data type determined by the function and expresses the truth value of the logical expression using a value of this type.

Note

These functions can be viewed as a partial replacement for the Boolean data type for truth values not available in ABAP. In particular, **xsdbool** and (with restrictions) **boolc** can be used in many operand positions where input parameters of the type **abap_bool** of the type group ABAP are expected.

Variant 1

```
... boolc( log_exp ) ...
```

Effect

The function **boolc** returns a single-character character string of the type **string**. If the logical expression is true, "X" is returned. If the logical expression is false, a blank is returned. In principle, **boolc** is one of the processing functions with character-like results and can be specified in general expression positions and in character-like expression positions.

Notes

- If **boolc** requires return values other than "X" or " " (for example, "Y" and "N" or "1" and "0"), the result of **boolc** can be edited directly using the function **translate** or another suitable processing function.
- The result of **boolc** must not be compared in relational expressions with the constants **abap_true** and **abap_false**, since the comparison converts the latter from **c** to **string** and ignores any blanks. Comparisons of this type are not usually necessary. If a comparison of this type is required anyway, the function **xsdbool** can be used instead of **boolc**. The result of this function has the same ABAP type as **abap_bool**.
- If the logical expression is false, the result of **boolc** does not meet the condition **IS INITIAL**, since a blank is returned (not an empty string). If this is the required behavior, the function **xsdbool** can be used instead of **boolc**.
- If **boolc** is used in inappropriate places (as specified in the points above), a syntax warning is produced (which can be hidden using a pragma).

Example

The value 0, 1, or 2 is assigned to the variable **bool_value**, depending on the result of the logical expressions **log_exp1** and **log_exp2**.

```
DATA bool_value TYPE i.
```

```
bool_value = strlen( condense( val = boolc( log_exp1 ) ) ) +
             strlen( condense( val = boolc( log_exp2 ) ) ).
```

Example

Calls a method, where the input parameter **no_dialog** is supplied with the character-like representation of the results of a predicate expression.

```
PARAMETERS word TYPE c length 30.
DATA result_tab TYPE cl_abap_docu=>search_results.

cl_abap_docu=>start(
  EXPORTING word           = word
            no_dialog      = boolc( sy-batch IS NOT INITIAL )
  IMPORTING search_results = result_tab ).
```

Variant 2

```
... boolx( bool = log_exp bit = bit ) ...
```

Effect

The function **boolx** returns a byte chain of the type **xstring**. If the logical expression is true, the byte chain is filled as if the function **bit-set(bit)** were being executed. If the logical expression is false, the byte chain is filled as if the function **bit-set(0)** were being executed. **bit** expects a data object of the type **i**. In principle, **boolx** is one of the bit functions and can be used in all positions where a bit expression is also allowed.

Note

The function **boolx** can be used for efficient saving of sequences of truth values.

Example

The result of the following bit expression is hexadecimal `55`, the same as the calculated bit string `01010101`.

```
DATA(result) = boolx( bool = 2 > 1 bit = 8 )
        BIT-OR boolx( bool = 2 < 1 bit = 7 )
        BIT-OR boolx( bool = 2 > 1 bit = 6 )
        BIT-OR boolx( bool = 2 < 1 bit = 5 )
        BIT-OR boolx( bool = 2 > 1 bit = 4 )
        BIT-OR boolx( bool = 2 < 1 bit = 3 )
        BIT-OR boolx( bool = 2 > 1 bit = 2 )
        BIT-OR boolx( bool = 2 < 1 bit = 1 ).
```

The bit expression above can be expressed using the following iteration with the operator **REDUCE**.

```
DATA(result) =
  REDUCE xstring( INIT x TYPE xstring
                  FOR j = 4 THEN j - 1 UNTIL j < 1
                  LET b1 = 2 * j b2 = 2 * j - 2 IN
                  NEXT x = x BIT-OR boolx( bool = 2 > 1  bit = b1 )
                             BIT-OR boolx( bool = 2 < 1  bit = b2 ) ).
```

Variant 3

```
... xsdbool( log_exp ) ...
```

Effect

Like **boolc**, the function **xsdbool** returns the value "X" for true and a blank for false. The data type of the return value, however, has the type **c** of the length 1 here.

The return value references the type XSDBOOLEAN from ABAP Dictionary. This type (which references the identically named domain with the type CHAR and length 1) is handled like a real Boolean type in serializations and deserializations

to or from asXML and asJSON using **CALL TRANSFORMATION**: The XML or JSON values `true` and `false` are assigned to the values "X" and " " of this type.

xsdbool can be specified in general and character-like expression positions.

Notes

- The result of **xsdbool** can be used like a value of the type **abap_bool** and compared with the constants **abap_true** and **abap_false**.
- If the logical expression is false, the result of **boolc** meets the condition **IS INITIAL**, since the returned blank is the type-friendly initial value too.
- The result of **xsdbool** cannot usually be implemented directly using a processing function such as **translate**, since the trailing blanks here are ignored in text fields with the type **c**. The result of a false logical expression would be ignored. The result of the function **boolc** with the type **string** is better suited to transformations of this type.
- The abbreviation **xsd** stands for XML schema data types.

Example

This example sets the type and the value of the variable **gui_flag** declared inline using the Boolean function **xsdbool**. A predicative method call is specified as the argument of this function. The variable is then serialized to asXML and asJSON using the predefined identity transformation ID. This produces the value `true` or `false`. After being compared with the identically typed constant **abap_false**, the result of the serializations is either read or displayed.

The result would be very different if **boolc** were used instead of **xsdbool**. Firstly, the transformations would have a different result (since the values "X" and " " are not transformed to `true` or `false`); secondly, the logical expression **gui_flag = abap_false** would always be false (since **abap_false** loses its blank when converted to the type **string**).

```
DATA(gui_flag) = xsdbool( cl_demo_sap_gui=>check( ) ).

CALL TRANSFORMATION id SOURCE gui_flag = gui_flag
                       RESULT XML DATA(xml).

DATA(writer) =
  cl_sxml_string_writer=>create( type = if_sxml=>co_xt_json ).
CALL TRANSFORMATION id SOURCE gui_flag = gui_flag
                       RESULT XML writer.
DATA(json) = writer->get_output( ).

cl_demo_output=>write_xml( xml ).
cl_demo_output=>write_json( json ).

IF gui_flag = abap_false.
  cl_demo_output=>get( ).
ELSE.
  cl_demo_output=>display( ).
ENDIF.
```