

## Dense ABAP Syntax Cheatsheet

```

DATA _var TYPE _type
DATA _var LIKE _var
CONSTANTS _var TYPE _type VALUE _value

DATA _data_ref TYPE REF TO _data_object
GET REFERENCE OF _var INTO _data_ref

CLASS _class DEFINITION
    _visibility SECTION
    INTERFACES _interface
    METHODS:
        _instance_method _parameter_interface
    CLASS-METHODS:
        _static_method _parameter_interface
    DATA _instance_attribute TYPE _type
    CLASS-DATA _static_attribute TYPE _type

CLASS _class IMPLEMENTATION
    METHOD _method

DATA _data_ref TYPE REF TO _class
CREATE OBJECT _data_ref TYPE _object's_class
Constructor: method named constructor

_visibility:
    PRIVATE, PROTECTED, PUBLIC

DEFINITION INHERITING FROM _superclass
„Overriding“:
    METHODS: _superclass_method REDEFINING

INTERFACE _interface
    METHODS:
        _interface_method

EVENTS _event
    EXPORTING VALUE(_ev_var) TYPE _type
*implement in class which defines the event:
    RAISE EVENT _event EXPORTING _ev_var = _var
*define as public in a class which reacts:
    METHODS: _ev_handler FOR EVENT _event OF
        _class_or_interface IMPORTING _ev_var
*during program execution define:
    SET HANDLER: _object->_ev_handler FOR _object

super->_superclass_method
_object->_instance_method_or_attribute
_class=>_static_method_or_attribute
_object->_interface~_interface_method

Method Class
ABSTRACT no implementation no objects
FINAL no overriding no inheritance
„STATIC“ shared by the whole tree N/A

```

```

_elementary_type: _custom_type:
i, f - num TYPES _type TYPE _type
c, string - char
d, t - date/time

Structures:
    TYPES: BEGIN OF _structure
            _component TYPE _type
            END OF _structure
Access structure: _structure-_component

_parameter_interface(classes and functions):
    IMPORTING _im_var TYPE _pi_type
    EXPORTING _ex_var TYPE _pi_type
    CHANGING _ch_var TYPE _pi_type
    RETURNING VALUE(_ret_var) TYPE _pi_type
_pi_type (special generics and others):
    ANY, ANY TABLE, INDEX TABLE, TABLE,
    STANDARD TABLE, SORTED TABLE, HASHED
    TABLE, _type

Parameters in interface can be:
    OPTIONAL, DEFAULT

CALL METHOD/CALL FUNCTION
    EXPORTING _im_var = _var (pass by value)
    IMPORTING _ex_var = _var (pass by value)
    CHANGING _ch_var = _var (pass by reference)
    RECEIVING _ret_var = _var (or functional call)

Functional call(functions as operands, only has
importing and returning):
    IF _functional( _var.._var )
Dynamic call:
    CALL METHOD _object->(_method_name)
    CALL FUNCTION _function_name
        Pass parameters with PARAMETER-TABLE
        WRITE _method_name/_function_name IN
        UPPERCASE LETTERS eg. 'OUTPUT_ME'

IF _expression WHILE _expression
ELSEIF _expression DO _number TIMES
ELSE _expression
=, <>, AND, NOT, OR and others

String operations:
    CONCATENATE _var.._var INTO _var
    CONDENSE _var NO-GAPS
    TRANSLATE _var TO UPPER CASE/USING _mask_pairs
    SEARCH _var FOR _var
    SPLIT _var AT _value INTO _var.._var
    STRLEN( _var )

```

```

Commenting: „inline
*whole line

(Internal) tables (like arrays, lists, queues):
    DATA _table TYPE _table_type
    TABLE OF _line_type WITH _key_type KEY _key

    TYPES _table TYPE _table_type OF
        _line_type WITH _key_type _key
_table_type:
    STANDARD, SORTED, HASHED
_line_type == _structure:
    DATA _line TYPE LINE OF _line_type
_key_type:
    UNIQUE, NON-UNIQUE

Table access:
    index, key

Table index operations(different with keys):
    READ TABLE _table INDEX _value INTO _line
    LOOP AT _table INTO _line
    APPEND _line TO _table
    INSERT _line INTO _table INDEX _value
    DELETE _table INDEX _value
    MODIFY _table FROM _line INDEX _value
    SORT _table
    CLEAR _table
*sum to existing one or append new entry
    COLLECT _line INTO _table

SQL query:
    SELECT _db_table~_db_column.._db_column
    FROM _db_table INNER JOIN _db_table ON
        _db_table~_db_column = _db_table~_db_column
    INTO (CORRESPONDING FIELDS OF) _table
    WHERE _db_column = _var.._db_column <> _var
    GROUP BY _db_column.._db_column
    HAVING _db_column = _var.._db_column <> _var
    ORDER BY _db_column ASCENDING/DESCENDING

WHERE additions:
    BETWEEN, LIKE, IN

Aggregate functions(use GROUP BY and HAVING):
    MAX, AVG, SUM, COUNT

Database cursor for iterative access:
    OPEN CURSOR _cursor FOR _SQL_query
    DO
        FETCH NEXT CURSOR _cursor INTO _line
        IF sy-subrc <> 0.
            CLOSE CURSOR. EXIT.

```

## Dense ABAP Syntax Cheatsheet

Native SQL:

```
EXEC SQL _native_statement
```

Files on application server:

```
OPEN DATASET _file_path FOR _operation  
IN _mode
```

```
TRANSFER _var TO _file_path  
READ DATASET _file_path INTO _var  
CLOSE DATASET _file_path
```

\_operation:

```
APPENDING, OUTPUT, INPUT
```

\_mode:

```
BINARY MODE, TEXT MODE
```

Field symbols(generic handling, pointers):

```
FIELD-SYMBOLS <_field_symbol> TYPE _type  
ASSIGN _var TO <_field_symbol>  
ASSIGN COMPONENT _component OF STRUCTURE  
_structure TO <_field_symbol>
```

Check existance:

```
_var IS INITIAL  
_data_ref IS BOUND  
_field_symbol IS ASSIGNED  
IN _table
```

Adressing subfields:

```
_var+_offset_value(_length_value)
```

Unit testing(inline comment must be written):

```
CLASS _t_class DEFINITION FOR TESTING. „#AU  
Risk_Level Harmless
```

METHODS:

```
_t_method FOR TESTING.  
CLASS _t_class IMPLEMENTATION.  
METHOD _t_method  
*execute function and other statements  
cl_aunit_assert=>assert_equals(  
act = _returned_result ,  
exp = _expected_result,  
msg = 'Display when false')
```

User memory (shared by all ABAP programs):

```
GET PARAMETER ID _field_id FIELD _var  
SET PARAMETER ID _field_id FIELD _var
```

ABAP memory (shared by a call sequence):

```
EXPORT _var TO MEMORY ID _value  
IMPORT _m_data = _var.._m_data = _var TO  
MEMORY ID _value  
DELETE FROM MEMORY ID _value
```

Math func: ABS, SIGN, CEIL, FLOOR, TRUNC, FRAC,  
all trigonometric, EXP, LOG, LOG10, SQRT

Generic and dynamic programming:

Get data object type as string:

```
DESCRIBE FIELD _var TYPE _s_var
```

New way(RTTS: RTTI, RTTC):

```
_t_var = cl_abap_typedescr=>  
describe_by_data( _var )  
_s_var = _t_var->get_relative_name()
```

Get structure components type names:

```
DATA _s_var TYPE REF TO cl_abap_structdescr  
_s_var ?= cl_abap_typedescr=>  
describe_by_data( _var )  
DATA _component TYPE abap_compdescr  
LOOP AT _s_var->components INTO _component  
_component-name, _component-type_kind, ...
```

Create and access data object dynamicaly:

```
DATA _var TYPE REF TO data  
CREATE DATA _var TYPE (_value)  
Access dynamicaly created object:  
FIELD-SYMBOLS <symbol> TYPE data  
ASSIGN _var->* TO <symbol>  
and from then on use <symbol>
```

```
MESSAGE _value TYPE _m_type
```

```
MESSAGE _t_nnn(_m_class)
```

```
_t_nnn:
```

```
_T is _m_type, _nnn are 3 digits in _m_class
```

```
_m_type:
```

```
'A', 'E', 'I', 'S', 'W', 'X'
```

\*if \_t\_nnn has & in definition, they are  
replaced with \_char..\_char:

```
MESSAGE _t_nnn(_m_class) WITH _chars.._chars
```

```
FORM _subroutine
```

```
USING _var (pass by reference)  
USING CHANGING _var (pass by reference)  
USING VALUE(_var) (pass by value)
```

```
PERFORM _subroutine
```

## Dense ABAP Syntax Cheatsheet

### System fields(flags with values):

SY-DATLO local date of the user  
SY-TIMLO local time of the user  
SY-INDEX current number of loop pass  
SY-TABIX last addressed table line  
SY-SUBRC return value of last run command  
SY-TCODE name of current transaction  
SY-UCOMM function code triggered during PAI  
SY-UNAME current user's name  
SY-REPID name of current ABAP program  
And many others

### Packages:

SABAPDEMOS : ABAP program examples  
SE83 : reuse library

### Function modules:

SPELL\_AMOUNT : currency to words  
C14W\_NUMBER\_CHAR\_CONVERSION : number to string  
HR\_HR\_LAST\_DAY\_OF\_MONTH : get the last day of  
the month

### Program:

RSTXSCRIP : SAPscript export/import to file  
RSUSR200 : lists data about user logons

SE02 : System Messages  
SE03 : Transport Organizer Tools (Excellent Doc)  
Change Object Directory Entries:Change Package  
SE09 : Transport Organizer  
SE11 : ABAP Dictionary  
SE14 : Database Utility (Detailed and Complex)  
SE15 : Repository (Search for Everything)  
SE16 : Data Browser (View and Create Entries)  
SE16n: General Table Display  
SE18 : ABAP Builder Definition  
SE19 : ABAP Builder Implementation  
SE24 : Class/Interface Builder  
SE30 : Runtime Analysis  
SE32 : Text Elements in Programs/Classes  
SE37 : Function Modules  
SE39 : Split Screen ABAP Editor  
SE41 : Menu Painter  
SE63 : Standard Translation Environment  
  
SE71 : SAPscript Form Painter  
SE72 : SAPscript Styles  
SE73 : SAPscript Font/Bar Code Maintenance  
SE75 : SAPscript Settings  
SE78 : SAPscript graphics  
  
SE80 : Object Navigator (Main Programming Tool)  
SE83 : Reuse Library  
SE90 : Transaction Maintenance  
SE91 : Message Maintenance  
SE92 : System Log Message Maintenance  
SE93 : Transaction Maintenance  
  
SM04 : User List  
SM31 : Table View Maintenance  
SM36 : Define Background Job  
SM37 : Execute Background Job  
SM49 : Execute Application Server Commands  
SM59 : RFC Connection Maintenance  
SM69 : Maintain Application Server Commands  
  
WE02 : IDoc List  
WE19 : Test Tool for IDoc  
WE20 : IDoc Communication Partner Profiles  
WE21 : Ports in IDoc  
WE30 : Develop IDoc Types  
WE31 : Define Segment Types

WE41 : Display Outbound Process Code  
WE42 : Display Inbound Process Code  
WE60 : IDoc Documentation  
WE81 : Display EDI:Logical Message Types  
WEDI : Enter A Special Menu  
  
AL11 : Application Server Directories  
BAPI : Business App Programming Interface  
BD64 : Distribution Model  
BD87 : Select IDoc, ALE Messages  
BSVW : Event Status Creation  
SWE2 : Event Type Linkage  
CG3Y : Download Files From Application Server  
CG3Z : Upload File To Application Server  
CMOD : SAP Enhancement Project Management  
SMOD : SAP Enhancement  
SECATT:Make and execute eCATTs  
SEW5 : Switch Framework, activate functions  
SHDB : Batch Input Transaction Recorder  
SMARTFORMS: Smart Forms Initial Screen  
SPRO : Customizing  
ST03N: Workload (User Activity by Transaction)  
ST22 : ABAP Runtime Error (View)  
STAD : Workload, Business Transaction Analysis  
STATTRACE: Functional Trace  
SU21 : Authorization Object Maintenance