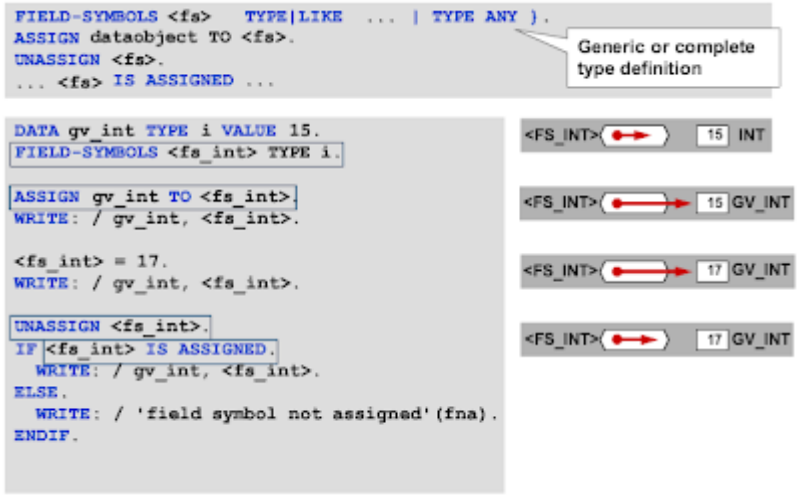
SapSystemsOfReza

TUESDAY, NOVEMBER 19, 2013

TAW12 - Dynamic Programming

FIELD SYMBOL



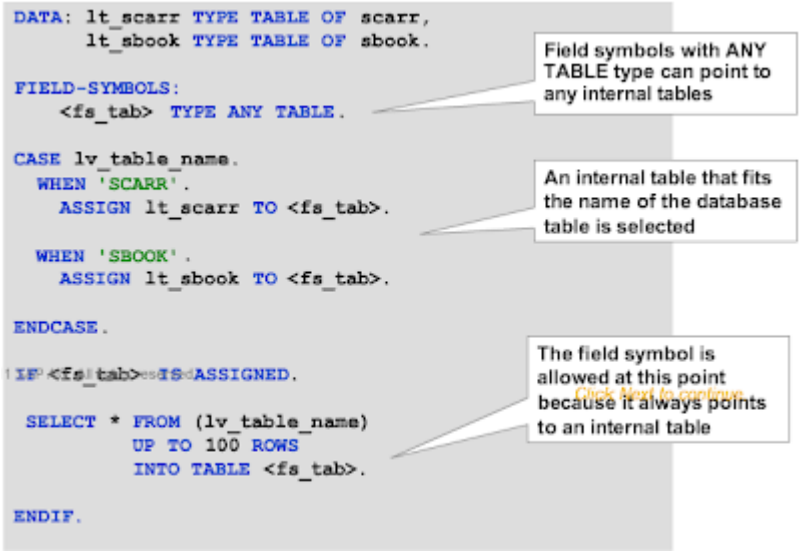
Field symbols are dereferenced pointers that have ‘symbolic’ access to a data object. Therefore, data object can be accessed using the field symbol itself. Field symbols are declared using the keyword, FIELD-SYMBOLS, as shown. Angular brackets (<>) are a part of the naming convention. The field symbols that are defined using TYPE or LIKE addition statically refer to a particular type. They can also be generically defined using TYPE ANY. You can assign a data object to a field symbol using the ASSIGN statement. As shown in the slide, the field symbol <fs_int> after assignment contains the value of gv_int. When the UNASSIGN statement is used, the field symbols do not

BLOG ARCHIVE

► 2019 (4)
► 2018 (7)
► 2016 (3)
► 2015 (1)
► 2014 (34)
▼ 2013 (19)
► December (1)
▼ November (15)
TAW12 - Object-Oriented Concept and Programming Te...
TAW 12 - INTRODUCTION TO OBJECT-ORIENTED PROGRAMMI...
TAW12 - Dynamic Programming
TAW10 - Screen Elements: Subscreen and Tabstrib Co...
TAW 10 - SCREEN ELEMENT FOR INPUT/OUTPUT
TAW 10 - SCREEN ELEMENT FOR OUTPUT
TAW10 - THE PROGRAM INTERFACE

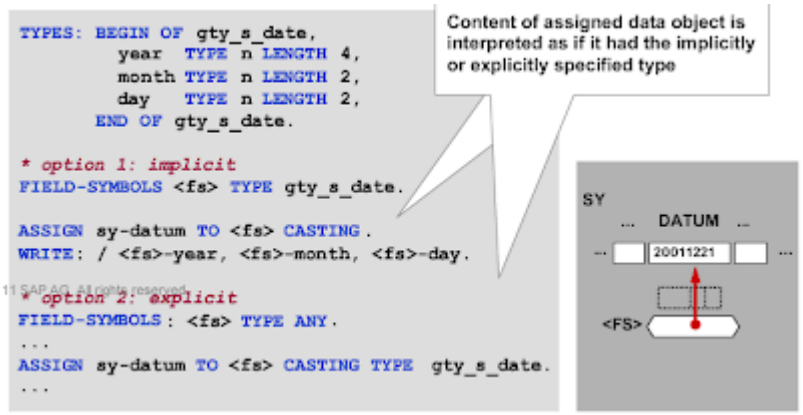
indicate any data object.

EXAMPLE: GENERIC FIELD SYMBOL IN DYNAMIC SELECT



If a field symbol is typed generically, it can be assigned to any data object irrespective of the type of data object. In this example,<fs_tab> is a field symbol of type internal table. Depending on the parameter supplied to the variable, lv_table_name, <fs_tab> should hold values from SCARR or SBOOK table. The SELECT statement is used to fill the internal table using <fs_tab>.

TYPE CASTING FOR FIELD SYMBOLS



TAW10 - Introduction to Screen Programming

TAW 10 - Search Help

TAW10 - views and maintenance dialog

TAW 10 - Changing Tables

TAW 10 - Object Dependencies - the repository info...

TAW 10 - Object Dependencies - Where-Used List

TAW10 - Object Dependencies - Runtime Objects

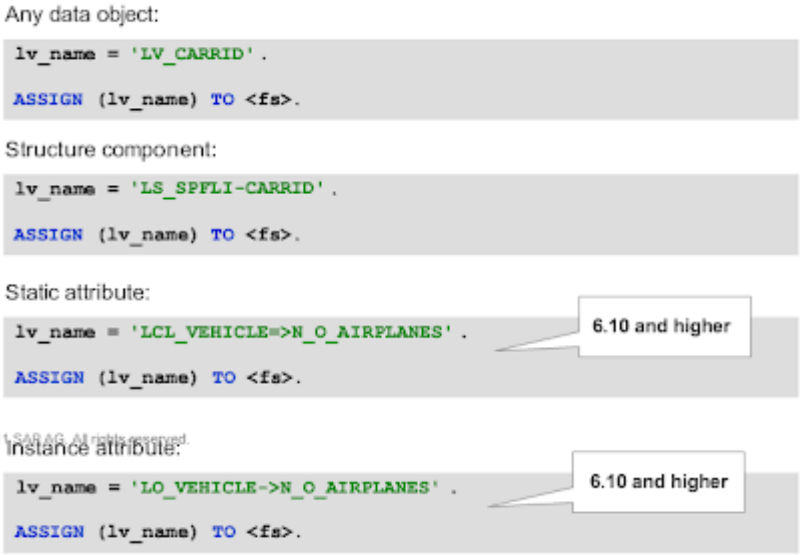
TAW 10 - Object Dependencies - active inactive ver...

► October (1)

► July (2)

A field symbol assigned to a data object can be used to refer to a different data object using the CASTING addition in the ASSIGN statement. The data object then behaves as if it implicitly had the data type of the field symbol. However, if the CASTING TYPE addition is used in the ASSIGN statement, then the data object behaves as if it is explicitly specified with the type of the field symbol. After the first ASSIGN statement, the field symbol <fs> will hold current date as shown here.

DYNAMIC ACCESS TO DATA OBJECTS



As shown in the slide, any data object, including structure components, can be assigned to a field symbol. After the release of version 6.1, even static and instance attributes can be assigned to a field symbol.

DYNAMIC ACCESS TO OBJECT ATTRIBUTE AND CLASS ATTRIBUTE

Static attribute (using the full name):

```
lv_name = 'LCL_VEHICLE=>N_O_AIRPLANES'.

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

Instance attribute:

```
lv_attribut_name = 'MAKE'.

ASSIGN lo_vehicle->(lv_attribut_name) TO <fs>.
```

Attribute name in uppercase or lowercase.

Static attribute:

```
lv_attribut_name = 'N_O_AIRPLANES'.
lv_class_name    = 'LCL_VEHICLE'.
```



```
ASSIGN (lv_class_name)=>(lv_attribut_name) TO <fs>.
```

Attribute name and class name in uppercase or lowercase.

Like data objects or structure components, dynamic access to the static attributes of a class or an instance of a class, can be made using field symbols, as shown here. The attribute name can be specified either in uppercase or lowercase. Even a class name can be mentioned either ways.

DYNAMIC ACCESS TO STRUCTURE COMPONENTS

Structure component (using the full name):

```
lv_name = 'LS_SPFLI-CARRID'.

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

Structure component (using its component name):

```
lv_comp_name = 'CARRID'.

ASSIGN COMPONENT lv_comp_name
    OF STRUCTURE ls_spfli     TO <fs>.
```

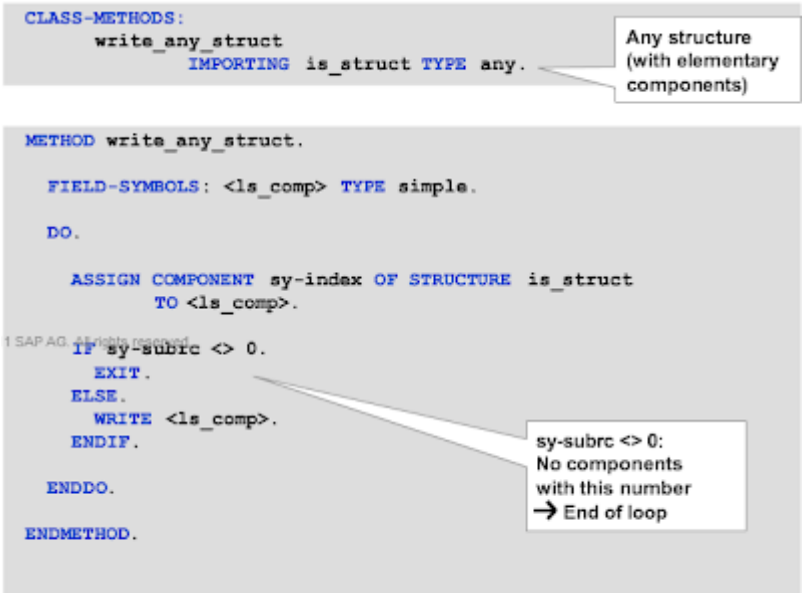
Structure component (using its position in the structure):

```
lv_comp_number = 2.

ASSIGN COMPONENT lv_comp_number
    OF STRUCTURE ls_spfli     TO <fs>.
```

We already know that structure components can be assigned to a field symbol. Using the ASSIGN statement, it is possible to access a particular component of a structure. As shown here, the syntax used for this type of assignment is ASSIGN COMPONENT

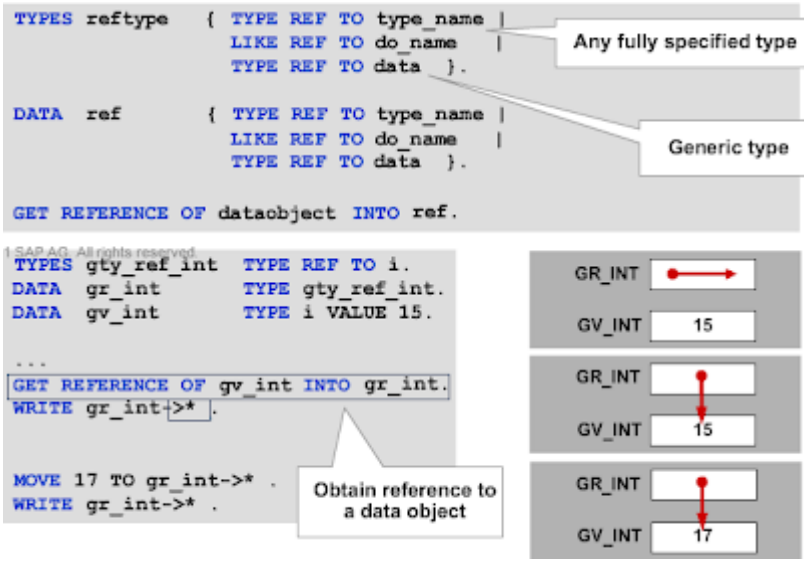
variable_name OF STRUCTURE. The component of a structure can also be accessed using its position.

FULL PROCESSING OF ANY NON-NESTED, FLAT STRUCTURE



The components of a structure can be accessed using its field symbols. Let us understand this with the example shown in the slide. The variable, is_struct, is an elementary structure declared as TYPE any. The system variable, sy-index, is used to assign the components of the structure to the field symbol, ls_comp.
The condition sy-subrc <> 0 (not equal to zero) checks the ASSIGN statement. If the condition is true, control exits the loop. Otherwise, the index value is displayed using the WRITE statement.

DATA REFERENCES



The reference type for a reference variable is defined using the TYPE REF TO addition in the TYPES statement. It can be either fully specified or generic. The DATA statement is used when you wish to assign a data object to a reference variable that is already defined with a specified type.
Let us understand the two statements with an example.
The reference variable gty_ref_int is of the type, Integer. The data object gr_int is also of the same type, as it refers to get_ref_int. The data object GV_INT holds a value 15.
By using the statement, GET REFERENCE OF, gr_int can point to gv_int. The dereferencing operator (→ *) is used to directly access the content of the data object.

VALIDATING REFERENCE VARIABLE: LOGICAL

Validity of References – Logical Expression

... ref IS [NOT] BOUND ...

The expression ref IS [NOT] BOUND is used to query whether the reference variable ref contains a valid reference. ref must be a data or object reference variable.

When a data reference is involved, this logical expression is true if it can be dereferenced. When an object reference is involved, it is true if it points to an object. The logical expression is always false if ref contains a null reference.

In contrast, you can only use the expression ... ref IS [NOT] INITIAL ... to determine whether ref contains the null reference or not.

The statement, ref IS [NOT] BOUND, is used to check if a reference variable contains a

valid reference. In the above statement, ref can be a data object or reference variable. If an object reference is involved, the logical expression returns “true” as the value. Otherwise, the value is “false.”

GENERIC DATA REFERENCES

- Parameter table (internal table) for dynamic procedure call
- During the dynamic generation of data objects using the CREATE DATA statement (see next lesson).

There are two options for addressing the contents of the referenced data object for references with type TYPE REF TO DATA. We discuss both of them below.

Reference variables that are generic can be created using TYPE REF TO DATA assignment. It is not possible to directly dereference a generically typed reference variable. Such reference variables can be used during dynamic internal table call and for dynamic creation of data objects.

CAST ASSIGNMENT FOR DATA REFERENCES

```
MOVE ref1 ?TO ref2.
```

```
DATA:
  lv_int  TYPE i VALUE 15,
  lv_date TYPE d VALUE '20040101'.

DATA:
  lr_int  TYPE REF TO i,
  lr_date TYPE REF TO d,
  lr_gen  TYPE REF TO data.
...
GET REFERENCE OF  lv_int INTO lr_int.
* up cast:
lr_gen = lr_int .
* down cast:
lr_int ?= lr_gen .

* but:
GET REFERENCE OF  lv_date INTO lr_date
* up cast:
lr_gen = lr_date .
* down cast:
lr_int ?= lr_gen .
```

Type fully specified

Generic type

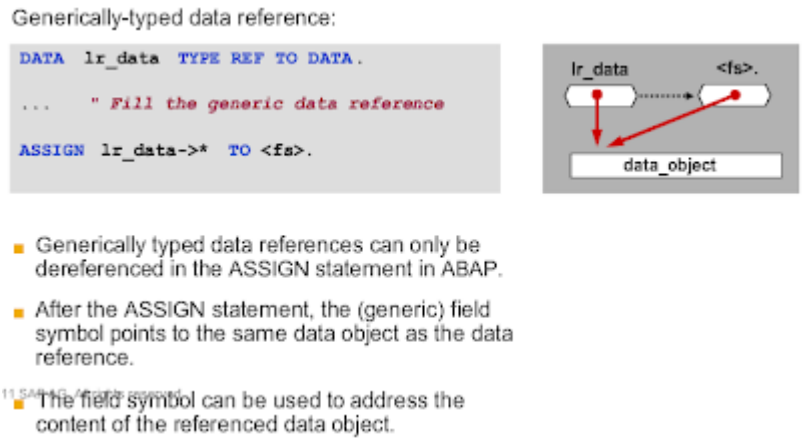
No type conflict

Type conflict!
Runtime error if not caught

When values are assigned between two reference variables of different types, it can be defined as cast assignment. As shown in the slide, lr_int and lr_date are fully typed reference variables; whereas, lr_gen is generically typed. When a generically typed reference variable is assigned to a fully typed reference variable, it is known as down cast. However, the opposite assignment is called up cast.

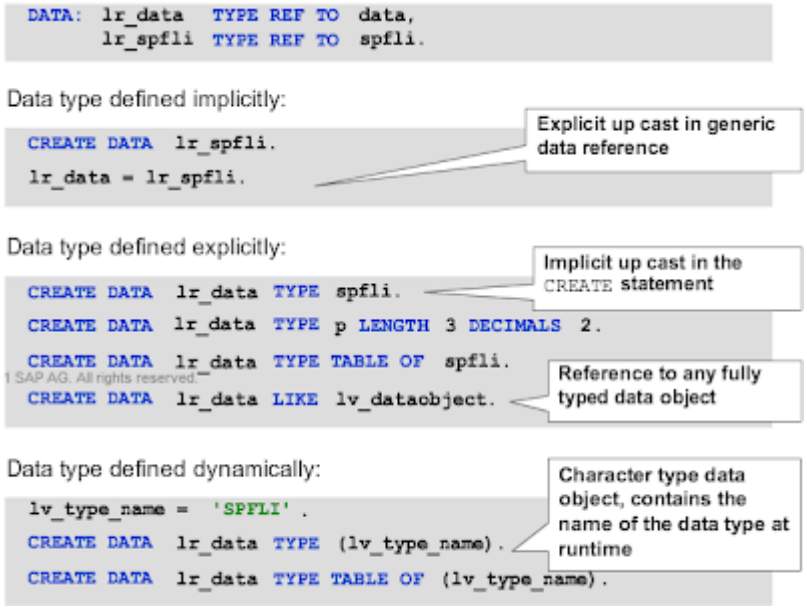
As seen here, down cast in the first case is type compatible. Since the assignment is not type compatible in the second case, execution of the program will lead to runtime error.

DEFERENCING GENERICALLY TYPED DATA



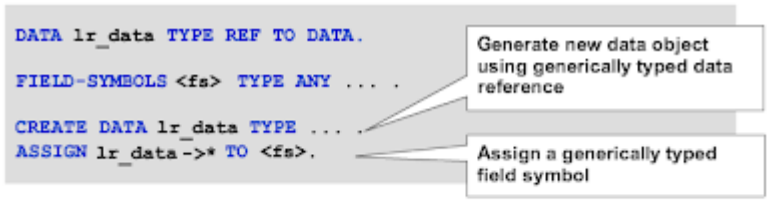
You can dereference generically typed reference variables only by using the ASSIGN statement. As shown in the slide, lr_data is assigned to the field symbol <fs> using the ASSIGN statement. After assignment, the field symbol points to the same data object as the data reference. Value of the data object reference can be obtained using the field symbol.

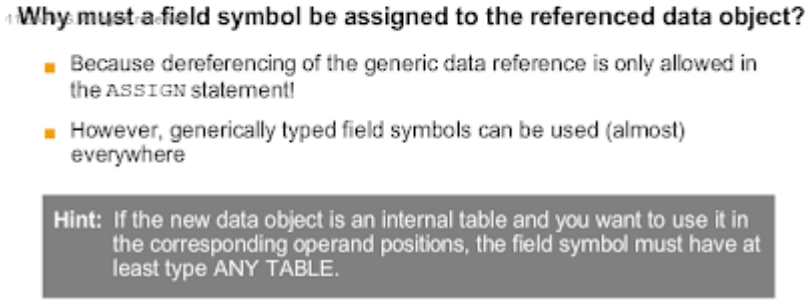
OPTIONS FOR CREATING DATA OBJECT AT RUNTIME



A data object pointing to a general data reference variable can be created dynamically using the CREATE DATA statement. As shown in the slide, when lr_spfli is assigned to lr_data, an up-cast is explicitly defined. However, the TYPE addition in a CREATE DATA statement implicitly generates an up-cast assignment. Using the LIKE addition in a CREATE DATA statement, references to an already existing data object can be specified. References to a character-type data object can be obtained by specifying the name of the object in parentheses in the CREATE DATA statement as shown here.

ACCESS TO DYNAMICALLY GENERATED DATA





As shown in the slide, CREATE DATA lr_data creates a data object whose type is unknown. Therefore, in case of dynamically created data objects, reference variable should be generic in nature. Using the ASSIGN statement, the field symbol <fs> points to the content of the data object, lr_data.
A field symbol is required in this case because

- Only the ASSIGN statement can be used to dereference a reference variable.
- Generically typed field symbols can be used anywhere.

EXAMPLE: DYNAMICALLY GENERATED INTERNAL TABLE AND DYNAMIC SELECT

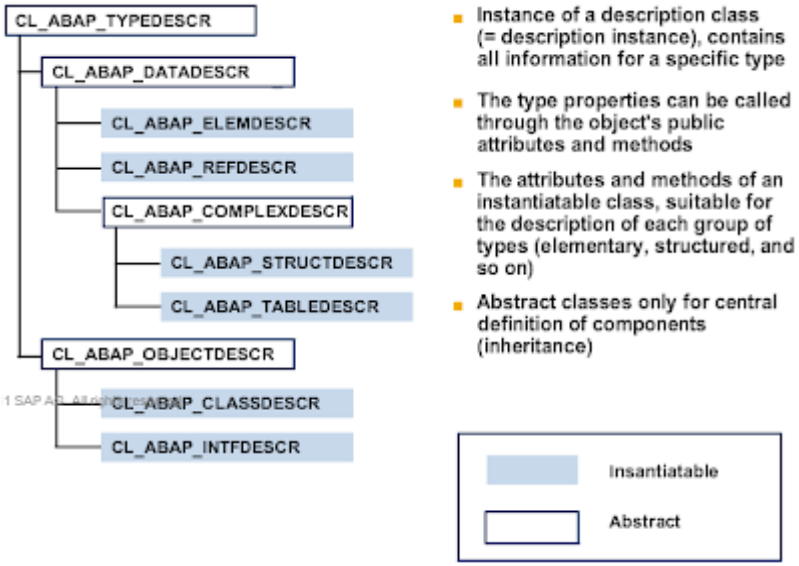
```
DATA: lv_tabname  TYPE        string,
      lv_max_rows TYPE        i,
      lr_table    TYPE REF TO data.

FIELD-SYMBOLS:<lt_table>  TYPE ANY TABLE.

lv_tabname  = 'SPFLI'.
lv_max_rows = 100.

CREATE DATA lr_table
       TYPE TABLE OF (lv_tabname).

ASSIGN lr_table->* TO <lt_table>.

SELECT * FROM (lv_tabname)
         UP TO lv_max_rows ROWS
         INTO TABLE <lt_table>.
```

Dynamic generation of an internal table with line type matching for DB table

Assignment of generically typed field symbol for usage in the SELECT statement

An internal table can be created dynamically using the TYPE TABLE OF addition in the CREATE DATA statement. The line type of the internal table is defined by specifying the name of the table in parentheses, as shown in the slide. In order to display the values, you can use a generically typed field symbol in the ASSIGN statement.

CLASS HIERARCHY OF RTTI DESCRIPTION CLASSES



- Instance of a description class (= description instance), contains all information for a specific type
- The type properties can be called through the object's public attributes and methods
- The attributes and methods of an instantiatable class, suitable for the description of each group of types (elementary, structured, and so on)
- Abstract classes only for central definition of components (inheritance)

ABAP defines a hierarchical list of global classes that can be used. An instance of the

class describes all the properties associated with the class which can be called using the public attributes and methods. Each class describes a specific category of types, for example CL_ABAP_TABLEDESCR is used to describe table types. The list is a combination of abstract classes used for inheritance, as well as those classes which can be instantiated.

RTTI CLASSES: INSTATIATION POSSIBLE

CL_ABAP_ELEMDESCR

To describe elementary data types

CL_ABAP_REFDESCR

To describe reference types (=types of reference variables)

CL_ABAP_STRUCTDESCR

To describe structure types

CL_ABAP_TABLEDESCR

To describe table types

CL_ABAP_CLASSDESCR

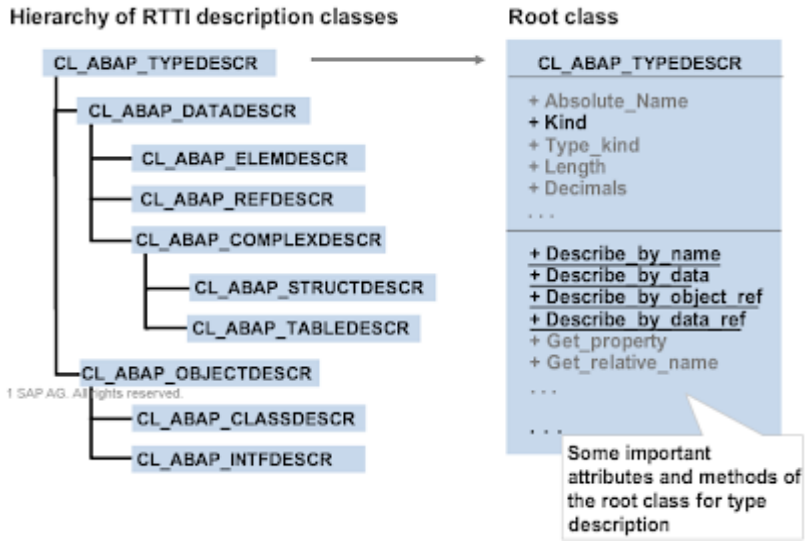
To describe classes (=object types)

CL_ABAP_INTFDESCR

To describe interfaces

Only six of the ten RTTI classes can be instantiated and used to describe specific types. All the other classes are abstract. They cannot be instantiated. This is because they are used to centrally define the attributes and methods in several other classes (and implement them if necessary). The METHODS attribute, which contains a list of the methods, is not defined in class CL_ABAP_CLASSDESCR. It is defined in class CL_ABAP_OBJECTDESCR, as it is also needed in the same form in class CL_ABAP_INTFDESCR.

RTTI: METHOD AND ATTRIBUTES OF THE ROOT CLASS



CL_ABAP_TYPEDESCR is the root class in the hierarchy of RTTI description classes. CREATE OBJECT cannot be used to instantiate class directly. As shown in the slide, you need to call the static methods, Describe_By_name and Describe_By_data of class CL_ABAP_TYPEDESCR to create an object reference.

DESCRIBING A TYPE BASED ON ITS NAME

```
DATA lo_type TYPE REF TO cl_abap_typedescr.
```

Analysis of a local data type

```
TYPES lty_type TYPE ....

lo_type = cl_abap_typedescr=>describe_by_name( 'LTY_TYPE' ).
```

Analysis of a global data type

```
lo_type = cl_abap_typedescr=>describe_by_name( 'SPFLI' ).
```

Analysis of a local object type (for example, local class)

```
CLASS lcl_class DEFINITION.
  ...
ENDCLASS.
lo_type = cl_abap_typedescr=>describe_by_name( 'LCL_CLASS' ).
```


Analysis of a global object type (for example, global interface)

```
lo_type = cl_abap_typedescr=>describe_by_name( 'IF_PARTNERS' ).
```

In order to access the attributes and methods of the respective type, you need to suitably cast the subclass with local or global data type. It is also possible to cast interfaces to the data objects.

CASING A SUITABLE REFERENCE FOR A TYPE DESCRIPTION OBJECT

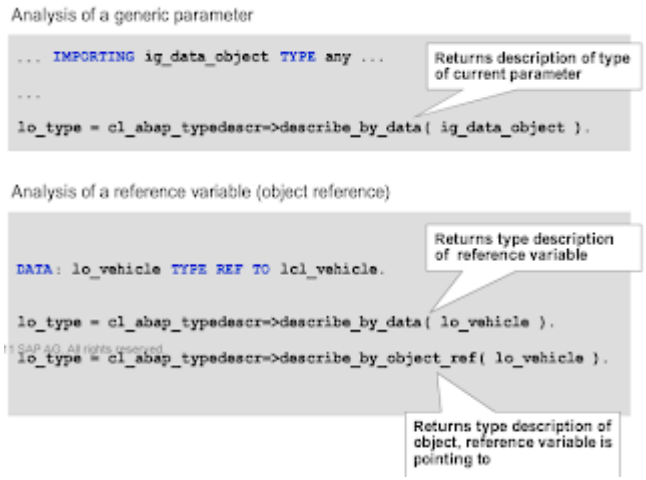
```
DATA:lo_type   TYPE REF TO cl_abap_typedescr,
     lo_elem   TYPE REF TO cl_abap_elemdescr,
     lo_ref    TYPE REF TO cl_abap_refdescr,
     lo_struct TYPE REF TO cl_abap_structdescr,
     lo_table  TYPE REF TO cl_abap_tabledescr,
     lo_intf   TYPE REF TO cl_abap_intfdescr,
     lo_class  TYPE REF TO cl_abap_classdescr.
```

```
lo_type = cl_abap_typedescr=>describe_by_ ... .

CASE lo_type->kind.
  WHEN cl_abap_typedescr=>kind_elem.
    lo_elem ?= lo_type.
  WHEN cl_abap_typedescr=>kind_ref.
    lo_ref ?= lo_type.
  WHEN cl_abap_typedescr=>kind_struct.
    lo_struct ?= lo_type.
  WHEN cl_abap_typedescr=>kind_table.
    lo_table ?= lo_type.
  WHEN cl_abap_typedescr=>kind_intf.
    lo_intf ?= lo_type.
  WHEN cl_abap_typedescr=>kind_class.
    lo_class ?= lo_type.
ENDCASE .
```

Evaluation of KIND (child) attribute, then down cast to correct subclass

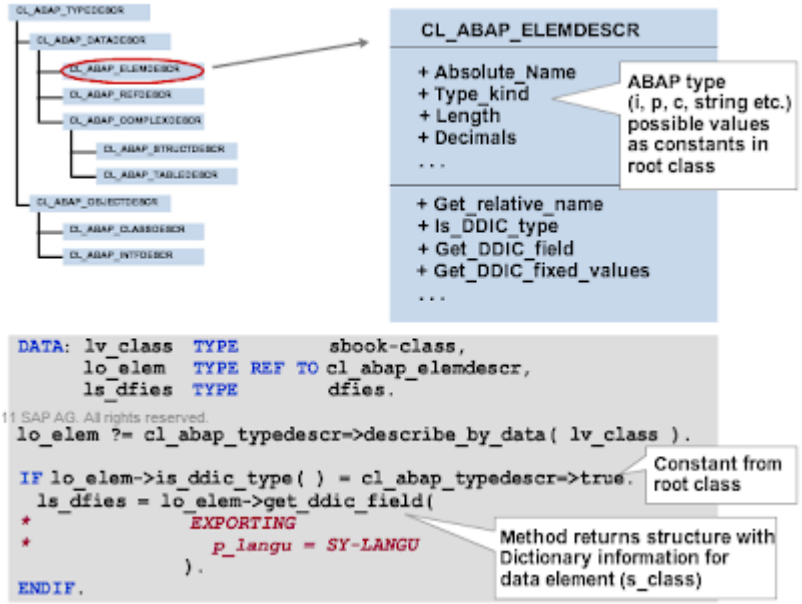
If the user has no idea about the RTTI class that was instantiated, then the public instance attribute, kind, can be used. However, a downcast to a suitable subclass must be done.

DESCRIBING TYPES BASED ON DATA OBJECTS AND REFERENCES



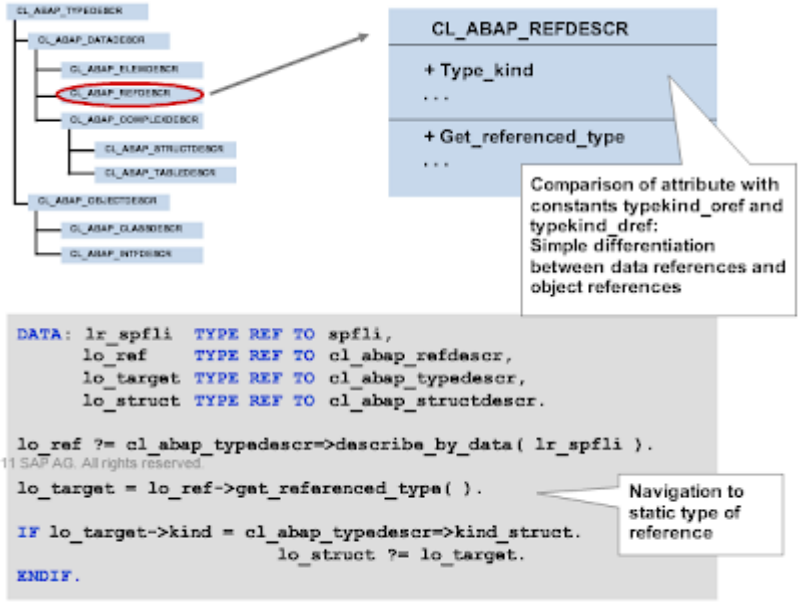
As shown in the slide, static methods can be used to describe different reference types. The method, describe_by_data, returns description about the type of current parameter in the first example whereas, the same method returns type description of the reference variable in the second example. Similarly, the method, describe_by_object_ref, gives the type description of the object that is pointed to by the reference variable.

ANALYSIS OF AN ELEMENTARY DATA TYPE



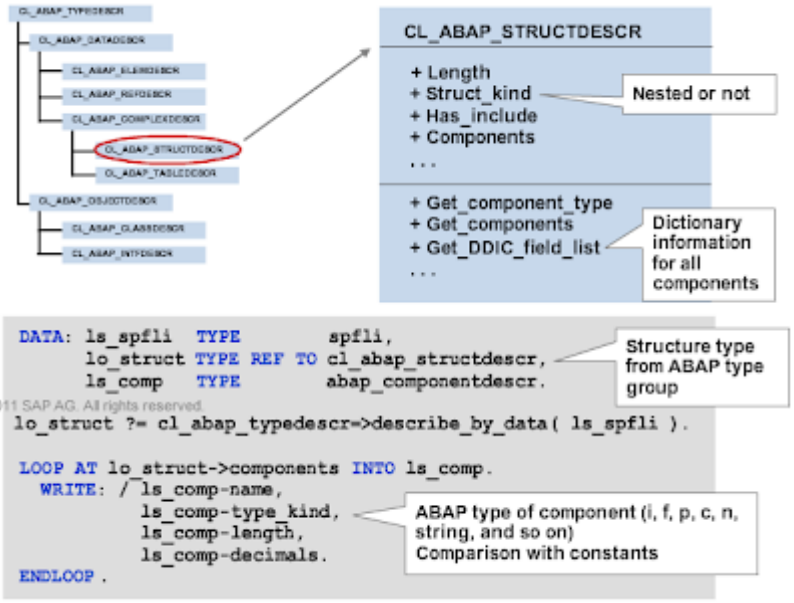
All the properties of an elementary data type can be accessed using the RTTI class, CL_ABAP_ELEMDESCR. Technical properties, such as type, length, and decimal places are defined in the respective public attributes. The public attribute, Type_kind, stores constants, which must be evaluated against the corresponding constant for the root class. The method, Get_DDIC_field, returns the structure with dictionary information for the data element.

ANALYSIS OF THE TYPE OF A REFERENCE VARIABLE



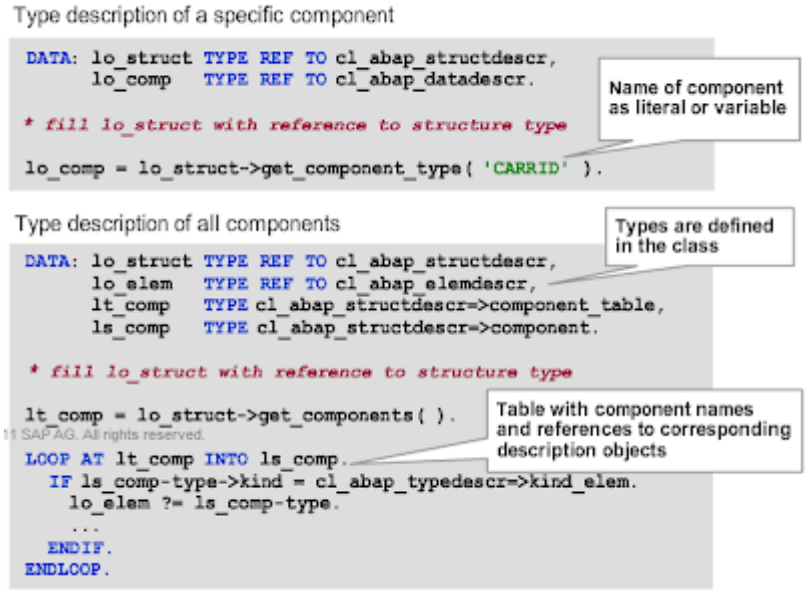
The RTTI class, CL_ABAP_REFDESCR, can be used to obtain the type of a reference variable. The inherited public attribute differentiates between data references and object references. The method, get_referenced_type, is used to describe the details of the static type of the reference variable.

ANALYSIS OF A STRUCTURE TYPE



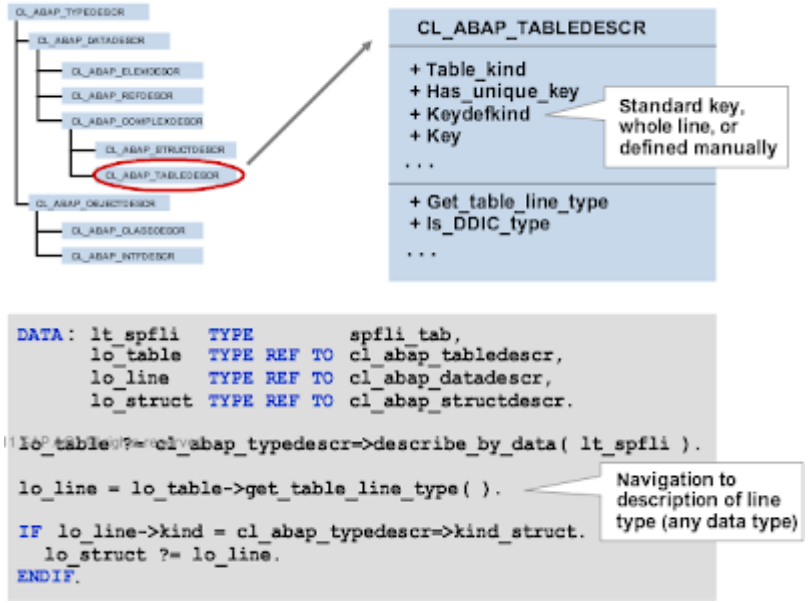
The class, CL_ABAP_STRUCTDESCR, is used to determine the structure of the reference variable at runtime. It contains public attributes, such as Length, Struct_Kind, Has_include, and Components. The attribute, struct_kind, determines whether the structure is nested or elementary. Similarly, the Components attribute is an internal table, which contains the names of all components and their technical properties. The method, Get_DDIC_field_list, describes the semantic information of all the components.

NAVIGATION FROM STRUCTURE TYPE TO COMPONENT TYPES



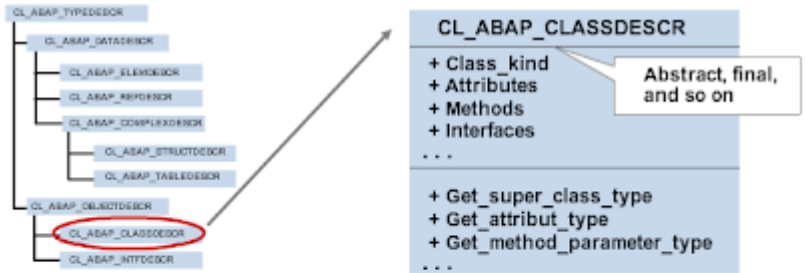
As shown in the slide, the method, get_componenet_type, returns the description object for the specified component. The type of reference variable, lo_elem, is identified by the class cl_abap_elemdescr. Using the variable, ls_comp, in the LOOP AT statement, description objects for all the components of the structure can be obtained.

ANALYSIS OF A TABLE TYPE



The RTTI class, CL_ABAP_TABLEDESCR, is used to determine the table type of the reference variable at runtime. Its public attributes contain information, such as whether the internal table is standard or sorted; and whether it has a unique key. The method, get_table_line_type, can be used to get the line type of the description object.

ANALYSIS OF AN OBJECT TYPE




```
DATA: lo_class     TYPE REF TO cl_abap_classdescr,
      ls_metdescr  TYPE        abap_methdescr,
      ls_parmdescr TYPE        abap_parmdescr.

lo_class ?= cl_abap_typedescr=>describe_by_name( 'CL_RENTAL' ).

READ TABLE lo_class->methods INTO ls_metdescr
           WITH KEY name = 'CONSTRUCTOR' .

LOOP AT ls_metdescr-parameters INTO ls_parmdescr.
  WRITE: / ls_parmdescr-name,
           ls_parmdescr-type_kind,
           ls_parmdescr-length.
ENDLOOP.
```

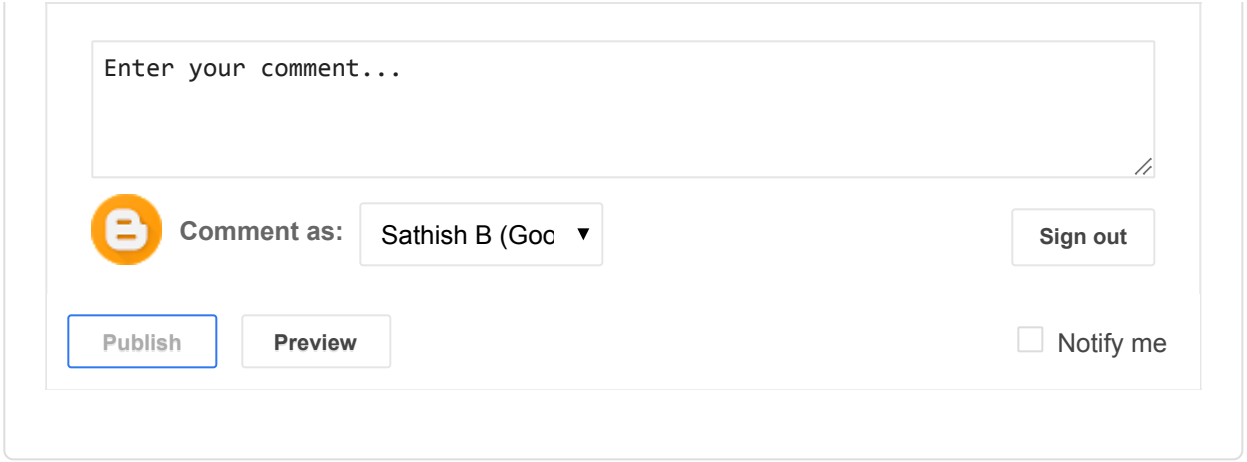
"Methods" attribute is a nested table with internal tables für parameters, exceptions, and so on

The public attributes of class CL_ABAP_CLASSDESCR contain information about attributes, methods, and interfaces of the described class. The attribute, Class_kind, determines if the class is declared as abstract or final. The Methods attribute is defined as a nested internal table, which contains parameters and exceptions for the respective method.

Posted by Fachreza R at 11:27 PM

No comments:

Post a Comment



Newer Post Home Older Post

Subscribe to: Post Comments (Atom)