NEW FEATURES IN ABAP 7.4

"A Thorough Compilation of ABAP for HANA Syntax!"

— THE CENTER CRITICS

A Complete Compendium of ABAP for 7.4 and HANA

COMPILED BY ANTHONY CECCHINI

Table Of Contents

Introduction to ABAP 7.4

The technical innovations in SAP are coming in rapid succession. It should therefore not come as a surprise that even the ABAP language is undergoing transformations. A host of features have been introduced in the ABAP 7.40 release, and with the upcoming 7.50 release even more new constructs can be added to your ABAP tool-kit. While nothing has been taken away, this is to ensure backward compatibility, the rate of change continues to accelerate. To get a clear technical understanding of ABAP 7.4, I recommend you read the following links…

One tool I suggest you do add to your ABAP arsenal, and what this blog series is based on, is the book ABAP® to the Future by Paul hardy. I believe Paul has currently released a 2nd edition as well. You can find the book here.

This blog series, like Paul's book, will focus on the changes that came with version 7.4 by breaking the blog up into sections a developer would normally be interested in, like string processing, or conditional logic…etc. Since you and I are developers, we tend to spend a good bit of time accessing the database when developing ABAP programs, so let's begin this series here….

New Commands in OpenSQL for ABAP 7.4

CASE Statements In OPEN SQL Queries in ABAP 7.4

One of the new features of ABAP 7.4 is the ability to use CASE statements in Open SQL queries. The code below shows an example of this. In this example there is a field in a local structure named ERNAM, and it should be filled with the literals "NAME1", "NAME2", or "NAME3" respectively, depending on the contents of the database field AUART (DocType).

```
DATA: ls_vbak TYPE vbak,
     ld_vbeln LIKE vbak-vbeln.

PARAMETERS: p_vbeln like vbak-vbeln.

CONSTANTS: lc_name1(5) TYPE c VALUE 'name1',
         lc_name2(5) TYPE c VALUE 'name2',
         lc_name3(5) TYPE c VALUE 'name3'.

ld_vbeln = p_vbeln.

SELECT vbeln, vbtyp,
 CASE
   WHEN auart = 'ZAMA' THEN @lc_name1
   WHEN auart = 'ZACR' THEN @lc_name2
  ELSE @lc_name3
END AS ernam
FROM vbak
WHERE vbeln = @ld_vbeln
 INTO CORRESPONDING FIELDS of @ls_vbak.
ENDSELECT.
```

```
SELECT vbeln, vbtyp,
 CASE
   WHEN auart = 'ZAMA' THEN @lc_name1
   WHEN auart = 'ZACR' THEN @lc_name2
  ELSE @lc_name3
END AS ernam
FROM vbak
WHERE vbeln = @ld_vbeln
 INTO CORRESPONDING FIELDS of @ls_vbak.
ENDSELECT.
```

Please make note that you have to put an @ symbol in front of your ABAP variables (or constants) when using new features, such as CASE, in order to let the compiler know that you are not talking about a field in the database. (This is called "Escaping" the Host Variable). You also have to put commas between the fields you are bringing back from the database and put the INTO statement at the end. This is a result of a new "strict" syntax check that comes into force when the compiler notices you are using one of the new features. In this way, SAP can still being backward compatible.

Why did I use this as my first example? Surely you have used the CASE statement on data AFTER you have retrieved it. So what have we gained or even done by placing the CASE inside the SELECT? Well, what the CASE statement has allowed you to do is outsource the conditional logic to the database, as opposed to performing the CASE on the application server.

If you are unfamiliar with coding ABAP on HANA, the paradigm shift of pushing logic down into the SAP HANA database to be processed is new, but can result in huge performance improvements. While this example is not HANA specific, it does introduce you to the “concept of pushing code down” to the database layer. Something in the past we have been told to avoid. The new paradigm in ABAP is “Code-to-Data”. We will learn to create VALUE by optimizing the backend DBMS (HANA).

Performing Calculations within SQL Statements in ABAP 7.4

Another feature that is new to release 7.4 is the ability to perform arithmetical operations inside of SQL statements. Before 7.4 you had to select the data first, then you could perform calculations on it. This is best explained with an example. Let’s say we are selecting against table SFLIGHT. We want all rows for United Airlines connection id 941. For each row, we will add together the total occupied seats in Business Class and First Class, then we will multiply that by price and store the result in field paymentsum of our internal table.

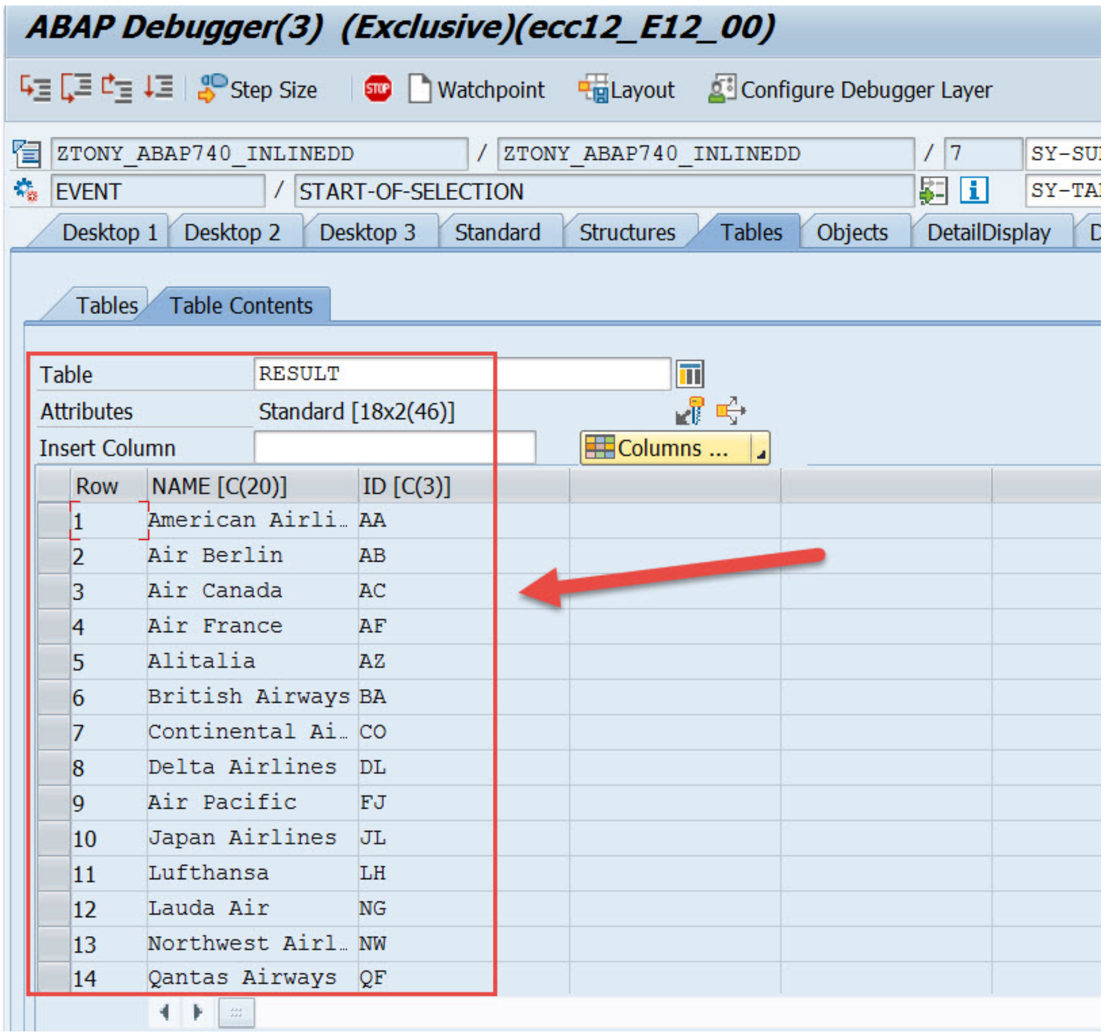
```
DATA: lt_sflight TYPE TABLE OF sflight.

 CONSTANTS: lc_carrid TYPE s_carr_id VALUE 'UA',
           lc_connid TYPE s_conn_id VALUE '941'.

 SELECT carrid, connid, price, seatsocc_b, seatsocc_f,
   ( ( seatsocc_b + seatsocc_f ) ) * price AS paymentsum
   FROM sflight
    WHERE carrid = @lc_carrid
      AND connid = @lc_connid
     INTO CORRESPONDING FIELDS of TABLE @lt_sflight.
```

In-Line Declarations within SQL Statements in ABAP 7.4

ABAP 7.4 has removed the need to create the data declaration for an internal table or structure. In prior versions of ABAP, if you declared a TYPE and then suddenly wanted to retrieve an extra field in your SELECT, then you would need to make the change in two places: in the TYPE definition and in the SELECT statement. In ABAP 7.4, however, you can not only skip the TYPE definition but the internal table declaration as well. An example of this is shown below:

```
SELECT carrname AS name, carrid AS id
    FROM   scarr
    INTO TABLE @DATA(result).
```

Look at the debugger screen shot below:



ZTONY_ABAP740_INLINEDD / ZTONY_ABAP740_INLINEDD / 7

EVENT / START-OF-SELECTION

Table	RESULT
Attributes	Standard [18x2(46)]
Insert Column	

Row	NAME [C(20)]	ID [C(3)]
1	American Airli…	AA
2	Air Berlin	AB
3	Air Canada	AC
4	Air France	AF
5	Alitalia	AZ
6	British Airways	BA
7	Continental Ai…	CO
8	Delta Airlines	DL
9	Air Pacific	FJ
10	Japan Airlines	JL
11	Lufthansa	LH
12	Lauda Air	NG
13	Northwest Airl…	NW
14	Qantas Airways	QF

As you can see, the table is created at the instant the database is accessed, and the format or ABAP TYPE of the table is taken from the types of the data fields you are retrieving.

This also works for structures if you are doing a SELECT on multiple database fields. The column name can also be influenced in the target internal table using the *AS <variable>* construct. So in the example below, in the internal table result, CARRNAME will be called NAME and CARRID will be called ID.

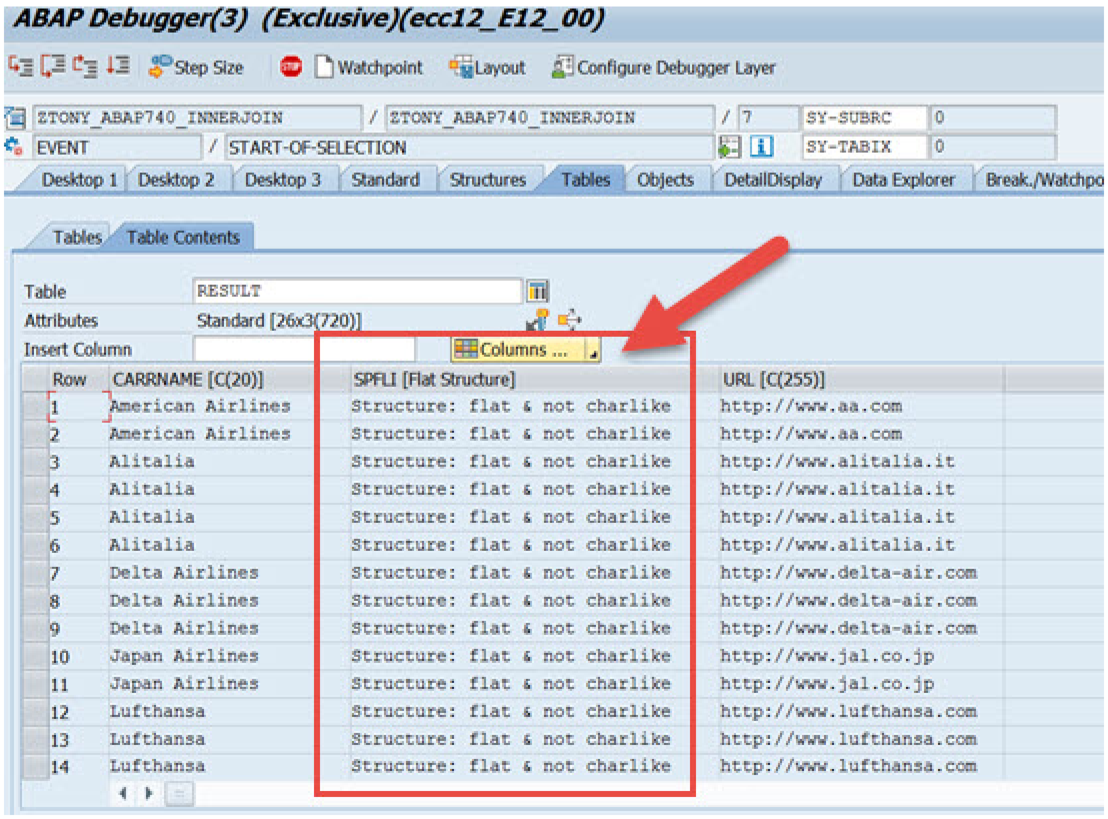
```
SELECT SINGLE carrname AS name, carrid AS id
      FROM   scarr
      WHERE  carrid = @id
      INTO @DATA(result).
```

INNER Join Column Specification in ABAP 7.4

As developers we (you) probably use Inner Joins frequently. In ABAP 7.4 we can utilize the ASTERISK in much the same way we can use it in a SELECT *. In the SELECT list, you can now specify all columns of a data source using the new syntax data_source~* (see below:)

```
SELECT scarr~carrname, spfli~*, scarr~url
       FROM scarr INNER JOIN spfli ON scarr~carrid = spfli~carrid
       INTO TABLE @DATA(result).
```

Take a look at the Debugger screen shot below:


ABAP Debugger(3) (Exclusive)(ecc12_E12_00)

ZTONY_ABAP740_INNERJOIN	/	ZTONY_ABAP740_INNERJOIN	/ 7	SY-SUBRC	0
EVENT	/	START-OF-SELECTION		SY-TABIX	0

Table RESULT
Attributes Standard [26x3(720)]
Insert Column

Row	CARRNAME [C(20)]	SPFLI [Flat Structure]	URL [C(255)]
1	American Airlines	Structure: flat & not charlike	http://www.aa.com
2	American Airlines	Structure: flat & not charlike	http://www.aa.com
3	Alitalia	Structure: flat & not charlike	http://www.alitalia.it
4	Alitalia	Structure: flat & not charlike	http://www.alitalia.it
5	Alitalia	Structure: flat & not charlike	http://www.alitalia.it
6	Alitalia	Structure: flat & not charlike	http://www.alitalia.it
7	Delta Airlines	Structure: flat & not charlike	http://www.delta-air.com
8	Delta Airlines	Structure: flat & not charlike	http://www.delta-air.com
9	Delta Airlines	Structure: flat & not charlike	http://www.delta-air.com
10	Japan Airlines	Structure: flat & not charlike	http://www.jal.co.jp
11	Japan Airlines	Structure: flat & not charlike	http://www.jal.co.jp
12	Lufthansa	Structure: flat & not charlike	http://www.lufthansa.com
13	Lufthansa	Structure: flat & not charlike	http://www.lufthansa.com
14	Lufthansa	Structure: flat & not charlike	http://www.lufthansa.com

You can see that SPFLI has been added to the table RESULT. Please remember to address the data for SPFLI you would need to code as follows…

```
RESULT[n]-SPFLI-data_element
```

Also, please, please, please… be mindful when using the asterisk. It acts just like the wild card in SELECT * and can impact performance if you really didn't want all of the columns.

Declaring and Creating Variables in ABAP 7.4

In this chapter, I'd like to introduce you to a new feature called Inline Declarations. It's a very simple feature, but very useful, and can make our code base smaller and easier to understand. Inline declaration means, that you declare your local variables as embedded in the given context, instead of declaring them separately at the beginning of program (i.e. the TOP INCLUDE).

ABAP 7.4 Data Type Declarations

The ABAP compiler knows what data type it wants, in fact it has to know in order to be able to perform a syntax check, so why not let it decide the data type of your variable and create it? So instead of declaring it yourself, let the compiler declare it for you.. lets look at some examples.

Before ABAP 7.4

```
DATA: lv_vehicle TYPE string.
lv_vehicle = 'Mercedes'.
```

What if instead, you could code this...

With ABAP 7.4

```
DATA(lv_vehicle) = 'Mercedes'.
```

OR this...

Before ABAP 7.4

```
DATA: lv_rows TYPE i.
lv_rows  = LINES( itab)
```

Becomes...

With ABAP 7.4

```
DATA(lv_rows) = LINES( itab ).
```

What about field symbols? Well For field symbols there is the new declaration operator FIELD-SYMBOL(…) that you can use now. Lets look at the 3 examples of how to use this new operator below…

With ABAP 7.4

```
ASSIGN ... TO FIELD-SYMBOL().

LOOP AT itab ASSIGNING FIELD-SYMBOL().
...
ENDLOOP.

READ TABLE itab ASSIGNING FIELD-SYMBOL() ...
```

Using The “NEW” Constructor Operator in ABAP 7.4

With Release 7.40 ABAP supports so called constructor operators. Constructor operators are used in constructor expressions to create a result that can be used at operand positions. The syntax for constructor expressions is

```
... operator type( ... ) ...
```

“NEW” is a constructor operator. “TYPE” is the explicit name of a data type. Inside the parentheses specific parameters can be specified.

So looking at some examples, for ABAP OO creating an instance …

Before ABAP 7.4

```
DATA lo_human TYPE REF TO class_human.
CREATE OBJECT lo_human EXPORTING NAME = 'TONY'.
```

With ABAP 7.4

```
lo_human = NEW class_human( name = ‘TONY’ ).
```

And for Data objects...

Before ABAP 7.4

```
DATA: lv_rows  TYPE i.
lv_rows  = 0.
```

With ABAP 7.4

```
lv_rows = NEW i(  0  ).
```

Using The “VALUE” Constructor Operator in ABAP 7.4

The VALUE constructor operator works similarly to the NEW Operator to create the ITAB entries. Using the VALUE operator, the itab would be initialized and records would be inserted. let’s look at an example, first we will create an table type...

```
TYPES t_itab TYPE STANDARD TABLE OF i WITH DEFAULT KEY.
```

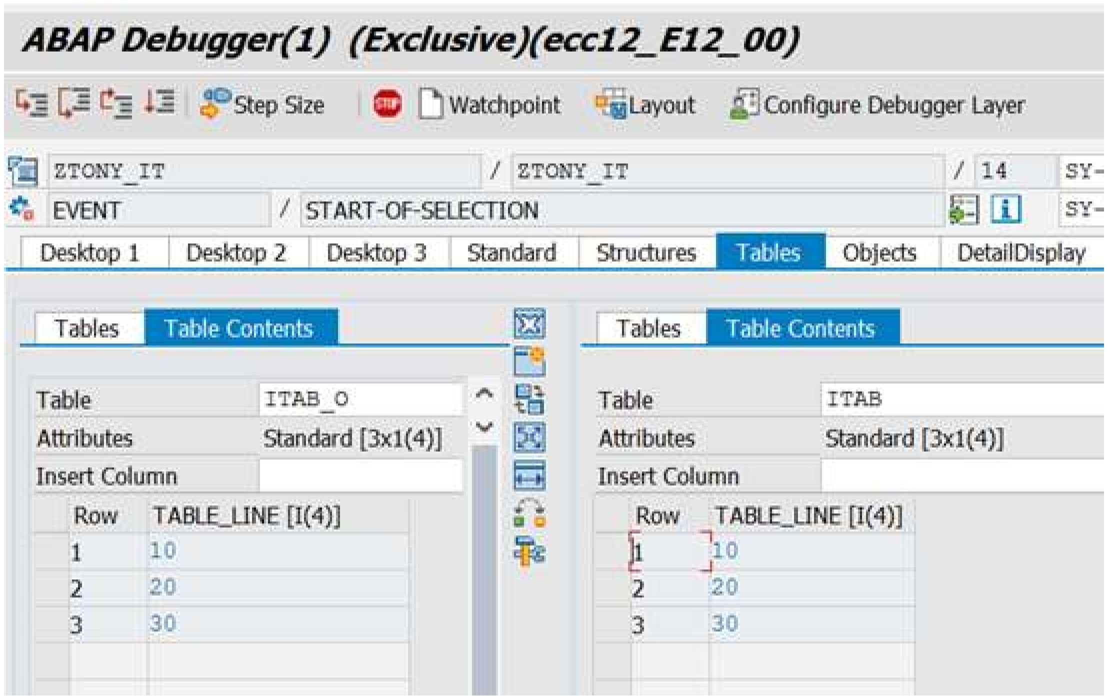
Before ABAP 7.4

```
DATA itab_o TYPE t_itab.
APPEND: 10 TO itab_o,
       20 TO itab_o,
       30 TO itab_o.
```

With ABAP 7.4

```
DATA(itab) = VALUE t_itab( ( 10 ) ( 20 ) ( 30 ) ).
```

OK, so let’s take a look at these inside the debugger. As you can see we accomplished the same goal with way fewer lines of code.


ABAP Debugger(1) (Exclusive)(ecc12_E12_00)

ZTONY_IT / ZTONY_IT / 14
EVENT / START-OF-SELECTION

Table	ITAB_O
Attributes	Standard [3x1(4)]
Insert Column	

Row	TABLE_LINE [I(4)]
1	10
2	20
3	30

Table	ITAB
Attributes	Standard [3x1(4)]
Insert Column	

Row	TABLE_LINE [I(4)]
1	10
2	20
3	30

Using the "FOR" Iteration Expression in ABAP 7.4

What is an *Iteration Expression*? Well you and I do not normally hard code our internal table entries as shown above. This is not feasible for large tables, and hard coding is generally frowned upon as a practice. Usually we fill our internal tables by reading the SAP database. We also filled one internal table from another, and could only do this if the columns were the same in the source and target internal tables. So prior to ABAP 7.4 you had to add all the lines of one table to another or do an assign as depicted below...

```
APPEND LINES OF lt_itab1 TO lt_itab2.
lt_itab2[] = lt_itab1[].
```

Now that the FOR command has been introduced in ABAP 7.4, you can achieve this in a much easier way, and the tables can have different columns, and you can filter or limit what gets transferred using conditional logic with the VALUE and FOR keywords. So what does FOR do? let's examine the syntax:

```
FOR wa| IN itab [INDEX INTO idx] [cond]
```

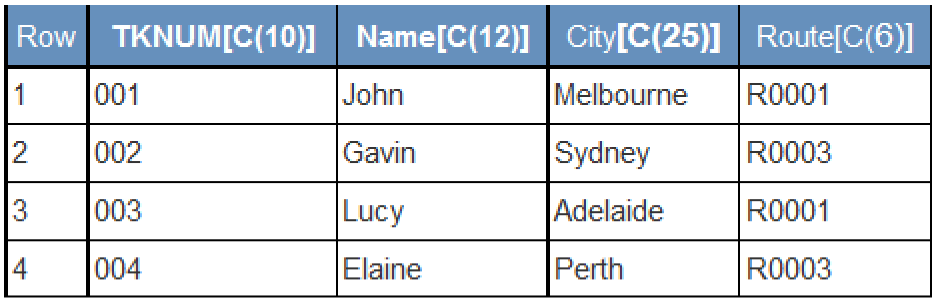
This effectively causes a loop at itab. For each loop the row read is assigned to a work area (wa) or field-symbol(). This wa or is local to the expression i.e. if declared in a subrourine the variable wa or is a local variable of that subroutine. Index like SY-TABIX in loop.

lets look an an example below...

Lets say we are given the following to start:

```
TYPES: BEGIN OF ty_ship,
         tknum TYPE tknum,     "Shipment Number
         name  TYPE ernam,     "Name of Person who Created the Object
         city  TYPE ort01,     "Starting city
         route TYPE route,     "Shipment route
       END OF ty_ship.
 TYPES: ty_ships TYPE SORTED TABLE OF ty_ship WITH UNIQUE KEY tknum.
 TYPES: ty_citys TYPE STANDARD TABLE OF ort01 WITH EMPTY KEY.
```

GT_SHIPS type ty_ships. -> has been populated as follows:



Row	TKNUM[C(10)]	Name[C(12)]	City[C(25)]	Route[C(6)]
1	001	John	Melbourne	R0001
2	002	Gavin	Sydney	R0003
3	003	Lucy	Adelaide	R0001
4	004	Elaine	Perth	R0003

We want to populate internal table GT_CITYS with the cities from GT_SHIPS.

Before ABAP 7.4

```
DATA: gt_citys TYPE ty_citys,
     gs_ship  TYPE ty_ship,
     gs_city  TYPE ort01.

LOOP AT gt_ships INTO gs_ship.
  gs_city =  gs_ship-city.
  APPEND gs_city TO gt_citys.
ENDLOOP.
```

With ABAP 7.4

```
DATA(gt_citys) = VALUE ty_citys( FOR ls_ship IN gt_ships ( ls_ship-city ) ).
```

OK, now lets throw some conditional logic into the mix. The goal now is to populate internal table GT_CITYS with the cities from GT_SHIPS where the route is R0001.

Before ABAP 7.4

```
DATA:  gt_citys TYPE ty_citys,
       gs_ship  TYPE ty_ship,
       gs_city  TYPE ort01.

LOOP AT gt_ships INTO gs_ship WHERE route = 'R0001'.
  gs_city =  gs_ship-city.
  APPEND gs_city TO gt_citys.
ENDLOOP.
```

With ABAP 7.4

```
DATA(gt_citys) = VALUE ty_citys( FOR ls_ship IN gt_ships
        WHERE ( route = 'R0001' ) ( ls_ship-city ) ).
```

Working with Strings in ABAP 7.4

SAP has added some interesting new String Functions in the ABAP 7.2. In some cases the new functions replace previous ABAP commands, and in other cases they bring some completely new functionality. Lets take a closer look...

Here are some of the more important changes to string processing in ABAP 7.2 and ABAP 7.4:

» Chaining Operator: chain two character-like operands into one new character string.

» String Templates: the option to create a character string out of literal texts, expressions, and control characters.

» Character String Functions: built-in functions

Using the Chaining Operator in ABAP 7.4

The Chaining Operator && can be used to create one character string out of multiple other strings and literals. The use of the chaining operator largely replaces the CONCATENATE statement. In this example, three variables are concatenated together using the && chaining operator.

```
DATA: v_var1 TYPE char30,
      v_var2 TYPE char30,
      v_var3 TYPE char30.
DATA: lv_result TYPE string.

v_var1 = 'Building'.
v_var2 = 'A'.
v_var3 = 'String'.

lv_result = v_var1 && v_var2 && v_var3.
WRITE: /(30) 'Using &&', lv_result
```

Using String Templates in ABAP 7.4

A string template is defined by using the | (pipe) symbol at the beginning and end of a template.

```
DATA: character_string TYPE string.
character_string = |This is a literal text.|.
WRITE: /(30) character_string.
```

The added value of a string template becomes clear when combining literal texts with embedded expressions and control characters. Embedded expressions are defined within a string template with curly brackets *{ expression }*. Note that a space between bracket and expression is obligatory. Some examples are:

```
character_string = |{ a_numeric_variable }|.
character_string = |This resulted in return code { sy-subrc }|.
```

Best of all, you can pass such constructs (the text between the starting | and the ending |) into parameters of method calls that are expecting strings. Previously, we had to create the string in a local variable and then use the local variable in the method call.

```
LO_OBJECT->STRING2XML( |{ converted_xml }{ xml_row-row_close_tag }| ).
```

Using Embedded Expressions in ABAP 7.2 and ABAP 7.4

In ABAP 7.2, the ALPHA formatting option was introduced and completely replaces the two function modules CONVERSION_EXIT_ALPHA_INPUT and CONVERSION_EXIT_ALPHA_OUTPUT. Now you can add or remove leading zeroes with this one .

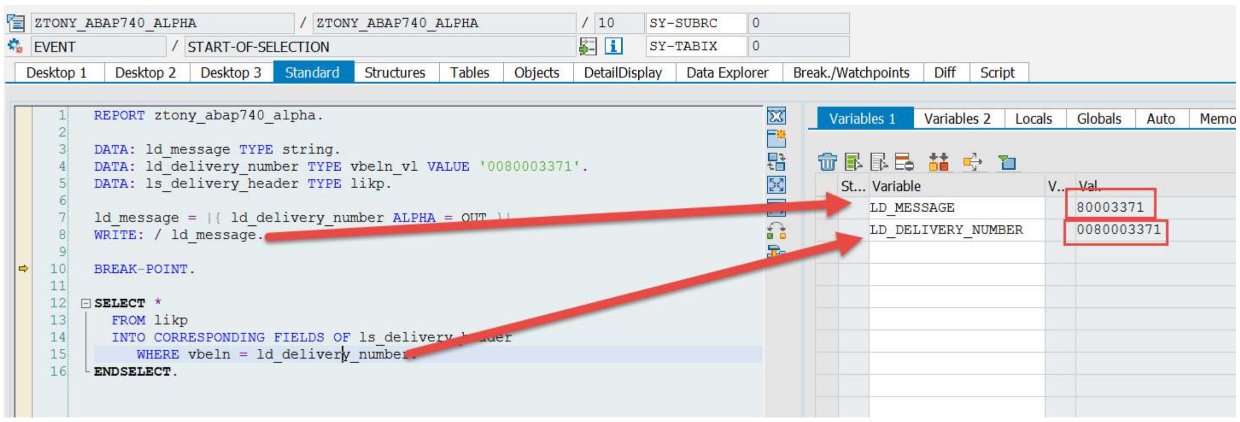
Below is the syntax for the ALPHA Embedded Expressions.

ALPHA = IN|OUT|RAW|(val)]

The parameter IN can be used to transform numeric sequences without leading zeroes to the format of numeric text with leading zeroes. The parameter OUT can be used to convert numeric text with leading zeroes to numeric sequences without leading zeroes. RAW is no formatting, and the possibilities of (val) are defined as constants in the class CL_ABAP_FORMAT. Lets look at an example…

```
DATA: ld_message TYPE string.
DATA: ld_delivery_number TYPE vbeln_vl VALUE '0080003371'.
DATA: ls_delivery_header TYPE likp.

ld_message = |{ ld_delivery_number ALPHA = OUT }|.
WRITE: / ld_message.

BREAK-POINT.

SELECT *
  FROM likp
  INTO CORRESPONDING FIELDS OF ls_delivery_header
    WHERE vbeln = ld_delivery_number.
ENDSELECT.
```

If we run this in the debugger, we can see that we no longer have to add the leading zeroes back—as they were never actually removed from the delivery variable in the first place.



```
REPORT ztony_abap740_alpha.

DATA: ld_message TYPE string.
DATA: ld_delivery_number TYPE vbeln_vl VALUE '0080003371'.
DATA: ls_delivery_header TYPE likp.

ld_message = |{ ld_delivery_number ALPHA = OUT }|.
WRITE: / ld_message.

BREAK-POINT.

SELECT *
  FROM likp
  INTO CORRESPONDING FIELDS OF ls_delivery_header
    WHERE vbeln = ld_delivery_number.
ENDSELECT.
```

St...	Variable	V...	Val.
	LD_MESSAGE		80003371
	LD_DELIVERY_NUMBER		0080003371

There are more Embedded Expressions than ALPHA in ABAP 7.2 and ABAP 7.4. I would recommend checking out the ABAP help , but here are a few more…

```
[WIDTH     = len]
[ALIGN     = LEFT|RIGHT|CENTER|(val)]
[PAD       = c]
[CASE      = RAW|UPPER|LOWER|(val)]
[SIGN      = LEFT|LEFTPLUS|LEFTSPACE|RIGHT|RIGHTPLUS|RIGHTSPACE|(val)]
[EXPONENT  = exp]
[DECIMALS  = dec]
[ZERO      = YES|NO|(val)]
[XSD       = YES|NO|(val)]
[STYLE     =  SIMPLE|SIGN_AS_POSTFIX|SCALE_PRESERVING
           |SCIENTIFIC|SCIENTIFIC_WITH_LEADING_ZERO
           |SCALE_PRESERVING_SCIENTIFIC|ENGINEERING
           |(val)]
[CURRENCY  = cur]
[NUMBER    = RAW|USER|ENVIRONMENT|(val)]
[DATE      = RAW|ISO|USER|ENVIRONMENT|(val)]
[TIME      = RAW|ISO|USER|ENVIRONMENT|(val)]
[TIMESTAMP = SPACE|ISO|USER|ENVIRONMENT|(val)]
[TIMEZONE  = tz]
[COUNTRY   = cty] ...
```

Calling Methods and Functions in ABAP 7.4

This chapter will discuss the new ABAP 7.4 functionalities that make calling functions and methods easier to code and easier to read. lets start with METHOD CHAINING.

Using Method Chaining in ABAP 7.4

You can now directly pass the returning value of one method call to a next method call without using any intermediate local variable. This is referred to as *Method Chaining.* Previously, we were only allowed to create a chain of statements using attributes of the class. Now you can include methods as well as attributes in a chained call. Method chaining is available since ABAP Release 7.0 EhP2.

Don’t get it confused with the chained statements, which You write using colon symbol colon symbol (:) .

We can directly pass the result of one method into the input parameter of another method without the need for an intermediate variable. Normally, we would declare a variable, fill it with the result of a method call, and pass that variable into another method. When using this new feature, we are able to reduce the amount of intermediate variables. This should improve code readability. Lets take an example.

```
CATCH zcx_exception INTO lo_exception.
lv_error_txt = lo_exception->get_error_msg( ).
zcl_my_screen_message=>display( im_error = lv_error_txt ).
```

By using *Method Chaining*, we can do away with having to declare the lv_error_txt variable by chaining the two method calls together.

```
CATCH zcx_exception INTO lo_exception.
zcl_my_screen_message=>display( im_error = lo_exception->get_error_msg( ) ).
```

Avoiding TYPE MISMATCH Errors in ABAP 7.4

We have all been there… you are developing an SAP ABAP program and you declare some local variables to be passed into a method, cross your fingers, and hope you have declared the variable the same as the input parameter. If not, you get an error. Worse, if it was a function module, you get a short dump. Well ABAP 7.4 can address at least the IMPORT side of the problem, using the new DATA TYPE declaration operator DATA(…).

Lets take an example…

```
DATA: ld_number_of_items TYPE i,
      ld_po_number       TYPE ebeln.

lo_purorder=>get_items( EXPORTING id_po_number = ld_po_number
                        IMPORTING ed_number_of_items = ld_number_of_items ).
```

With ABAP 7.4, we can accomplish the same thing by declaring the variables returned from the method **not at the start of the routine** but instead, at the instant they have their values filled by the method. Like so…

```
lo_purorder=>get_items( EXPORTING id_po_number = ld_po_number
                        IMPORTING ed_number_of_items = DATA( ld_number_of_items ) ).
```

What are the advantages of using this feature?
-> You cannot possibly get a type mismatch error or dump.
-> If you change the method signature definition or formal parameter type of a function module, then the code adapts itself accordingly, again avoiding the type mismatch.

Therefore, this coding approach is easier to maintain, and safer. This approach really starts to make sense when creating ABAP object instances using factory methods; let me show you what I mean…

Here is some ABAP code to create a Mercedes Benz subclass .

```
DATA: lo_vehicle TYPE REF TO zcl_mercedes_benz.
lo_vehicle = zcl_car_factory=>build_new_car( ).
```

Now instead of using the TYPE REF to decide what subclass lo_vehicle should be, let's let the Factory Method Decide the Exact Subclass.

```
DATA( lo_vehicle ) = zcl_car_factory=>build_new_car( ).
```

Constructor Operators in ABAP 7.4

What is a *Constructor Operator*? Well the SAP definition is "A constructor expression consists of a predefined constructor operator, a data type or object type that matches the operator and that can be derived implicitly from the operand position using #, and type-specified parameters specified in parentheses. Each constructor expression creates a result whose data type is determined using the specified type. The parameters specified in parentheses are used to pass input values".

With this is mind, lets take a look at a specific situation – Often the result of one FORM, METHOD, or FUNCTION has to have its type converted before it can be passed into another FORM, METHOD, or FUNCTION. Lets take an example where we need to pass a name to the get_as_string method. We have a field im_delivery that is type vbeln_vl yet a string is expected. CONV # takes car of the conversion in-line. We can use the # character as a symbol for the operand type because data type required in an operand position is unique and fully identifiable. If it wasn't then we would need to specify a non-generic data type.

```
lcl_text_reader( )->get_as_string( id     = '0002'
                                   name   = conv #( im_delivery )
                                   object = 'VBBK' ).
```

There a more Constructor Operators that just CONV, take a look at the SAP ABAP 7.4 help for more information.

Conditional Logic in ABAP 7.4

The continuing theme in all of these posts, is how we can make our code thinner, more readable and transparent. Well, nothing clogs up the visual readability of your code like long IF-THEN-ELSE-ENDIF or CASE-WHEN-ENDCASE constructs. Well, in ABAP 7.4 we have some new *Constructor Operators* we can use in *Constructor Expressions* that will make our code easier to read, more compact, and safer to maintain.

You remember *Constructor Operators*… We discussed them in the last blog post New Features in ABAP 7.4 – Calling Methods and Functions. In That post we highlighted the CONV *Constructor Operator* and how we could use it to convert a local variable to a STRING. In this post, I'd like to talk about the Constructor Operators COND and SWITCH. Again, for a complete list see the SAP ABAP 7.4 help for more information.

Using COND as a Replacement for IF/ELSE in ABAP 7.4

Since CASE statements can only evaluate one variable at a time, we use the IF/ELSE construct when we need to check multiple conditions. Look at the example below…

```
DATA: lv_text(30).

IF lv_vehicle = '01' AND lv_type = 'C'.
    lv_text = 'Toyota'.
ELSE.
IF lv_vehicle ='02' AND lv_type = 'C'.
   lv_text = 'Chevy'
ELSE.
IF lv_vehicle ='03' AND lv_type = 'C'.
   lv_text = 'Range Rover'.

  ..
ENDIF.
```

In ABAP 7.4, you can achieve the same thing, but you can do this in a more economical way by using the COND constructor operator. This also means that you do not have to keep specifying the target variable again and again. We will also add an ABAP inline declaration by using the DATA statement to create the variable lv_text inline on the fly!

```
DATA(lv_text) = COND text30(
    WHEN lv_vehicle ='01' AND lv_type = 'C' THEN 'Toyota'
    WHEN lv_vehicle ='02' AND lv_type = 'C' THEN 'Chevy'
    WHEN lv_vehicle ='03' AND lv_type = 'C' THEN 'Range Rover').
```

Using SWITCH a Replacement for CASE in ABAP 7.4

Here, we're getting the day of the week and using a CASE statement to turn the number into a string, such as "Monday", to output at the top of a report.

```
data: l_indicator like scal-indicator,
     l_day(10) type c.

call function 'DATE_COMPUTE_DAY'
    exporting
        date = p_date
    importing
        day = l_indicator.

case l_indicator.
    when 1.
      l_day = 'Monday'.
    when 2.
      l_day = 'Tuesday'.
    when 3.
      l_day = 'Wednesday'.
    when 4.
      l_day = 'Thursday'.
    when 5.
      l_day = 'Friday'.
    when 6.
      l_day = 'Saturday'.
    when 7.
      l_day = 'Sunday'.
else.
     Raise exception type zcx_day_problem.
endcase.
```

With ABAP 7.4, we can simplify and accomplish the same thing by by using the new SWITCH constructor operator instead of the CASE sttatement. Like so...

```
DATA(L_DAY) = SWITCH char10( l_indicator
  when 1 THEN 'Monday'
  when 2 THEN 'Tuesday'
  when 3 THEN 'Wednesday'
  when 4 THEN 'Thursday'
  when 5 THEN 'Friday'
  when 6 THEN 'Saturday'
  when 7 THEN 'Sunday'
    ELSE THROW zcx_day_problem( ) ).
```

Also note that if you are tired of only catching exceptions you can throw exceptions now! The usage is identical to the RAISE EXCEPTION TYPE, however, the compiler evaluates the keywords RAISE EXCEPTION TYPE and THROW as if they were one and the same.

Using Predictive Method Calls in ABAP 7.4

For quite some time the lack of a real Boolean type in ABAP led to a lot of discussions amongst developers. I would see post on the SCN community constantly asking why a developer couldn't write..

```
IF meth( ... ).
  ...
ENDIF.
```

The answer from SAP was always "Because you have to have a Boolean type behind IF and no method can return a Boolean type." But, now in Release 7.40, SP08 You can write predicative method calls as shown below..

```
... meth( ) ...
```

as a short form of the relational expression

```
... meth( ) IS NOT INITIAL ...
```

So you can place simple functional method calls everywhere, where logical expressions are allowed: behind IF, CHECK, ASSERT, COND, SWITCH, ... Here is an example...

```
IF zcl_system=>is_production( ).
"In production we never want a short dump
    TRY.
      zcl_application=>main( ).
      CATCH cx_sy_no_handler INTO gcl_no_handler.
    ENDTRY.
```

The reason this works is spelled out completely in Paul Hardy's Book ABAP to The Future. Paul explains, "What's happening from a technical point of view is that if you don't specify anything after a functional method the compiler evaluates it as IS_PRODUCTION( ) IS NOT INITIAL. An ABAP_TRUE value is really the letter X, so the result is not initial, and so the statement is resolved as true."

To learn more about Predictive Methods and Predicate methods, take a look at this Documentation and then check out Horst Keller's Blog.

Using the New Boolean Function XSDBOOL in ABAP 7.4

Another common situation with respect to Boolean logic in ABAP, is when you want to send a TRUE/FALSE value to a method or get such a value back from a functional method. For instance, did you ever stumble over this one?

```
IF boolc( 1 = 2 ) = ABAP_FALSE.
  WRITE: / 'YES'.
ELSE.
  WRITE: / 'NO'.
ENDIF.
```

Guess what? The relational expression "boolc(1 = 2) = ABAP_FALSE" is false! Why? Because BOOLC despite its name does not return c but a string and the comparison rules for c and string blah, blah, blah …

Technically BOOLC(1 = 2) evaluates to a string containing a blank of length 1. So far so good. But comparing ` ` with ' ' or ABAP_FALSE is false, since the text field is converted to string resulting in an empty string.

Now in ABAP 7.4, a new built-in function was added, called XSDBOOL, which does the same thing as BOOLC but returns an ABAP_BOOL type parameter.

```
IF xsdbool( 1 = 2 ) = ABAP_FALSE.
  WRITE: / 'YES'.
ELSE.
  WRITE: / 'NO'.
ENDIF.
```

The relational expression xsdbool(1 = 2) = abap_false is true, because xsdbool returns type XSDBOOLEAN from the ABAP Dictionary that is – yes, you guess it – c of length 1. For the experts among you, XSDBOOLEAN is normally used for special mappings in XML-transformations and was reused here for quite another purpose. And that's were the funny name xsdbool comes from. Again, I highly recommend reading Horst Keller's Blog on all the ABAP 7.4 enchantments to the code base.

Using Secondary Keys to Access Internal Tables in ABAP 7.4

All of us who have been developing in ABAP at one time or another have created a custom index on a database table. Why do we create a custom index or z-index? For performance… we recognize that a table could be queried in more ways then just by the primary key, so we setup customer indexes that we believe will be used by the Database Optimizer when determining the access path and thus make the query performant.

OK, back to internal tables, traditionally, if you wanted to read an internal table in two different ways (e.g.,looking for a material by Material Number or by Reference Number), then you either had to keep sorting the table just before a read, or have two identical tables sorted differently. Well now as of ABAP 7.2 can declare secondary keys for internal tables. The SAP Help states that using the secondary key could increases read access performance significantly. But, on the other hand, secondary keys also incur additional administration costs due to memory consumption and run-time.

For example, lets create a secondary index into the internal table IT_MARA for the column BISMT , this is just like having a secondary Z- index on BISMT in the database table definition. The internal table definition could be as shown below.

```
DATA: IT_MARA TYPE HASHED TABLE OF mara
          WITH UNIQUE KEY matnr
          WITH NON-UNIQUE SORTED KEY sort_key COMPONENTS bismt.
```

The SAP Help states that statements that previously only accessed the primary key have been enhanced so that access to secondary keys is now possible. Check out the help for a full list, but we will look at the READ TABLE statement here.

The code would look something like the below…

```
READ TABLE it_mara INTO wa_mara WITH KEY sort_key COMPONENTS bismt = lv_bismt.
```

Even though IT_MARA is a HASHED table, it is also a SORTED table with the key BISMT, so when we go looking for the record using BISMT a BINARY SEARCH is automatically performed.

Declaring Table Work Areas in ABAP 7.4

In release ABAP 7.4, the syntax for reading into a work area and looping through a table can now leverage INLINE DECLARATIONS, we discussed these in a prior ABAP 7.4 blog.

We learned that from 7.4 onward you no longer need to do a DATA declaration for elementary data types. It is exactly the same for the work areas, which are of course structures. Take a gander at the code below…

```
READ TABLE lt_mara WITH KEY matnr = lv_matnr INTO DATA(ls_mara).
LOOP AT lt_mara INTO DATA(ls_mara).
```

In the same way that you no longer need DATA declarations for table work areas, you also no longer need FIELD-SYMBOL declarations for the (common) situations in which you want to change the data in the work area while looping through an internal table. In ABAP 7.4, if you want to use field symbols for the work area, then the syntax is shown below…

```
READ TABLE lt_mara WITH KEY matnr = lv_matnr ASSIGNING FIELD-SYMBOL().
LOOP AT lt_mara ASSIGNING FIELD-SYMBOL().
```

Table Expressions in ABAP 7.4

What if I told you that you would never have to use the statement READ TABLE again to get a line out of an internal table?

That is exactly what happens in ABAP 7.4. Through the use of Table Expressions, a new way for accessing table lines in operand positions. We can view a table expression simply as a short form of a READ TABLE statement. The syntax for using a table expression consists of an internal table, followed by a row specified in square brackets [ ].

lets look at some code…

PRIOR to ABAP 7.4

```
READ TABLE flight_schedules INTO DATA(flight_schedule)
  WITH KEY carrid = 'AA'
           connid = '0017'.

lo_mara = zcl_mara_factory( ls_mara-matnr ).
```

ABAP 7.4

```
DATA(flight_schedule) = flight_schedules[ carrid = 'AA' connid = '0017' ].

lo_mara = zcl_mara_factory( lt_mara[ matnr = lv_matnr ]-matnr ).
```

The result of a table expression is a single table line. If a table line is not found, the exception CX_SY_ITAB_LINE_NOT_FOUND is raised. One way around this is to use the built-in table functions provided by SAP. One of them is the line_exists(…), which we can think as the short form of the READ TABLE … TRANSPORTING NO FIELDS statement. First, we check the existence of the specified row, and if it exists, then we perform the table read.

```
IF line_exists( vendors[ id = '00AED' ] ).
  vendors[ id = '00AED' ].
ENDIF.
```

From 7.40, SP08 on, we are able to define default values for avoiding the mentioned exception above. So, if the specified row is not found, then it returns the default value back.

```
DATA(default_customer) = VALUE customer( id = '00000' name = 'not found' ... ).

DATA(lv_customer) = VALUE #( customers[ id = '00024' ] DEFAULT default_customer ).
```

CORRESPONDING Constructor Operator in ABAP 7.4

The new constructor operator CORRESPONDING allows to execute a “MOVE-CORRESPONDING” for structures or internal tables at operand positions. Besides the automatic assigning components of the same name, you can also define your own mapping rules! This is best explained by using an example… We have have 2 internal tables LT_MARA_SOURCE and LT_MARA_TARGET. (1) You do not want to copy the MATKL value from one table to the other, even though both tables have a MATKL column. (2) You want to copy the column named KUNNR from one table into a similar column called CUSTOMER in the second table. We code use the code below, with the mapping rules defined…

```
lt_mara_source = CORRESPONDING #( lt_mara_target MAPPING customer = kunnr EXCEPT matkl ).
```

Here is the link to the SAP help.

MOVE-CORRESPONDING for Internal Tables

You can use MOVE-CORRESPONDING not only for structures but also for internal tables in ABAP 7.4. Components of the same name are are assigned row by row. New statement additions EXPANDING NESTED TABLES and KEEPING TARGET LINES allow to resolve tabular components of deep structures and to append lines instead of overwriting existing lines.

```
MOVE-CORRESPONDING itab_source TO itab_target EXPANDING NESTED TABLES
                                  KEEPING TARGET LINES.
```

I want to once again recommend that you grab a copy of Paul Hardy's book ABAP to The Future. He has done an excellent job explaining this specific concept in language developers can easily relate. Here is a link to the SAP help. (Not quite as easy on the eyes as Paul's book ;-))

Using The Constructor Operator FILTER in ABAP 7.4

The filter operator does what its name suggests it filters the content of an internal table. As a prerequisite, the filtered table must have a sorted or a hash key (primary or secondary), that is evaluated behind the WHERE clause.

```
DATA( lt_materials_fert ) = FILTER #( lt_all_materials USING KEY mtart WHERE mtart = 'FERT' ).
```

You can get quite fancy by adding some of the additional variants like EXCEPT and (ITAB ….IN …..ITAB) which behaves much like the FOR ALL ENTRIES does on the Database. Take a look at the SAP help for a more complete understanding.

Predicate Functions for Internal Tables in ABAP 7.4

A *predicate function* is predictive of a result. With ABAP 7.4 we have new built-in functions LINE_EXISTS and LINE_INDEX that fit this description for internal tables. Earlier in this blog I gave you an example of using LINE_EXISTS. Lets take a look at it again. We can think of using LINE_EXISTS as the short form of the READ TABLE … TRANSPORTING NO FIELDS statement. First, we check the existence of the specified row, and if it exists, then we perform the table read.

```
IF line_exists( vendors[ id = '00AED' ] ).
  vendors[ id = '00AED' ].
ENDIF.
```

Here is the SAP Help for LINE_EXISTS. Another *predicate function* is LINE_INDEX. In ABAP 7.4 release, we have new syntax LINE_INDEX() to identify the index of a row when a condition is met while reading the internal table. The new syntax is similar to READ TABLE with TRANSPORTING NO FIELDS followed by sy-subrc check. if sy-subrc = 0, then sy-tabix will give the index of the row.

```
READ TABLE it_schedule TRANSPORTING NO FIELDS
   WITH KEY carrid = 'US'
            connid = '24'.
IF sy-subrc = 0.
  WRITE: sy-tabix. "index
ENDIF.
```

Here is the SAP Help for LINE_INDEX.

Search Helps in ABAP 7.4

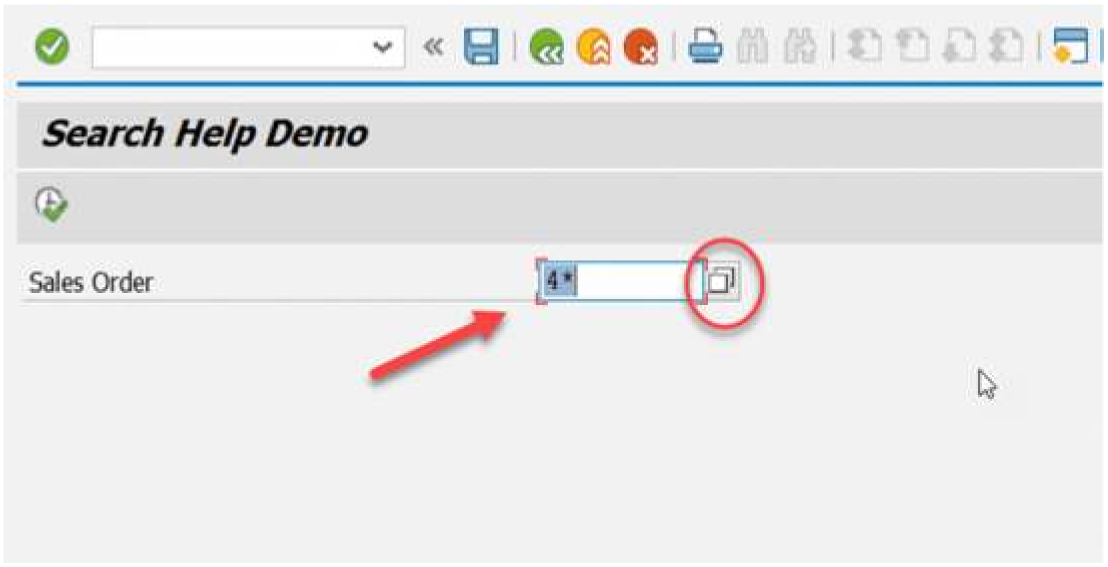
At some point in your ABAP career, you probably will develop a search help for a screen field to aid the user in entering correct data. In the "Classic" days (Everything old in SAP is Classic) these were called Match Codes.This blog post is not a tutorial on how to create a search help, please see the SAP ABAP 7.4 help for that. Instead, I am going to introduce you to some new functionality that became available in ABAP 7.4.

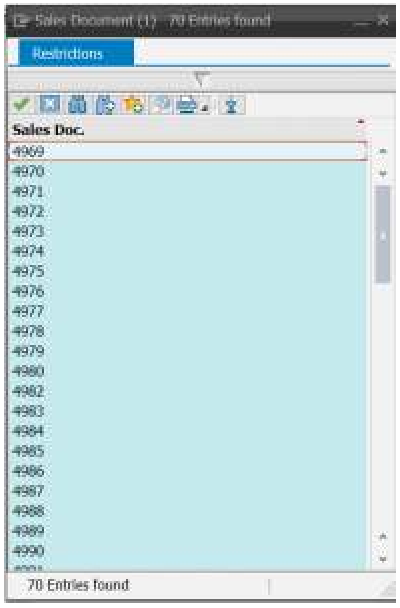
First, let's quickly review what a search help is...

A Search Help, a repository object of ABAP Dictionary, is used to display all the possible values for a field in the form of a list. This list is also known as a hit list. You can select the values that are to be entered in the fields from this hit list instead of manually entering the value, which is tedious and error prone.

There are several types of Search helps:
Elementary search helps: This type implements a search path for determining the possible entries.
Collective search helps: This type contains several elementary search helps. A collective search help, therefore, provides several alternative search paths for possible entries.
Append search helps: This type can be used to enhance collective search helps delivered by SAP with customer-specific search paths without requiring a modification.

An example of an elementary search help is shown below. You will see the icon next to the field. You enter a pattern and hit this icon or F4 and the hit list is displayed for you to choose from. (see below).

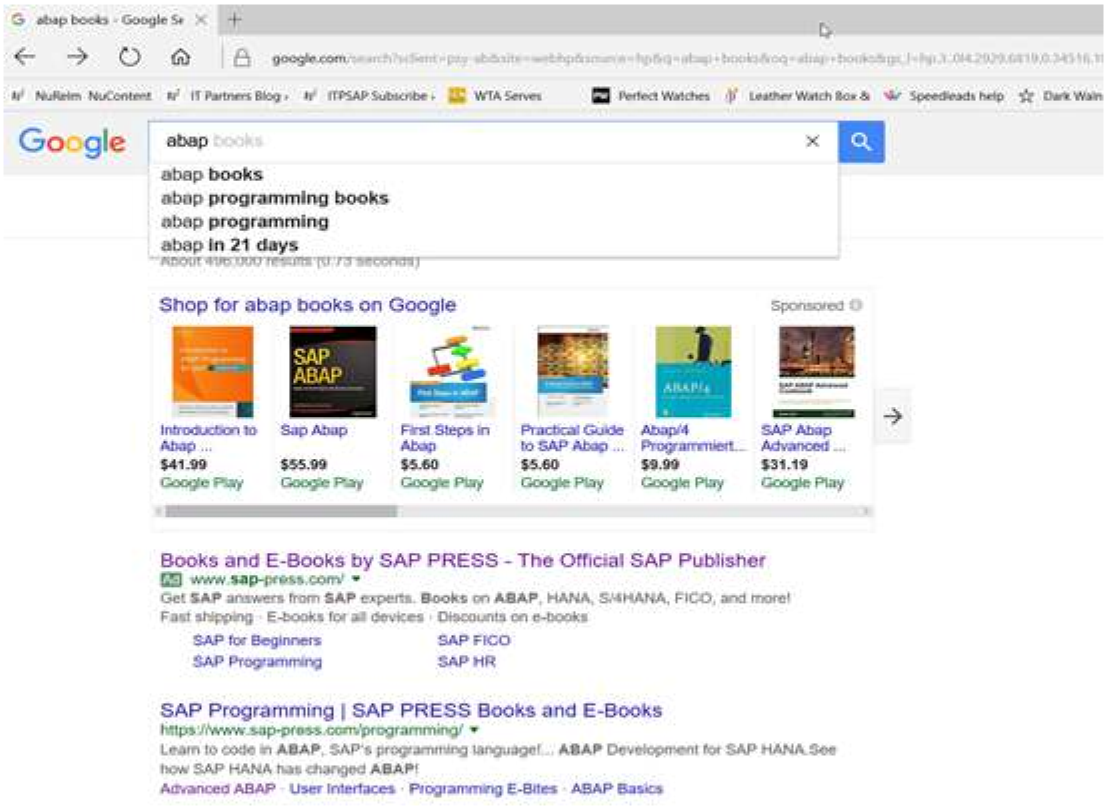



Sales Document (1) 70 Entries found

```
Sales Doc.
4969
4970
4971
4972
4973
4974
4975
4976
4977
4978
4979
4980
4982
4983
4984
4985
4986
4987
4988
4989
4990
```

70 Entries found

So you want to find a book on ABAP, so you head over to your favorite browser and using Google start typing in ABAP and instantly you see search results. (see below)



abap books

abap **books**
abap **programming books**
abap **programming**
abap **in 21 days**

Shop for abap books on Google

Introduction to Abap ...	Sap Abap	First Steps in Abap	Practical Guide to SAP Abap ...	Abap/4 Programmiert...	SAP Abap Advanced ...
$41.99	$55.99	$5.60	$5.60	$9.99	$31.19
Google Play	Google Play	Google Play	Google Play	Google Play	Google Play

Books and E-Books by SAP PRESS - The Official SAP Publisher
Ad www.**sap**-press.com/
Get **SAP** answers from **SAP** experts. **Books** on **ABAP**, HANA, S/4HANA, FICO, and more!
Fast shipping · E-books for all devices · Discounts on e-books
SAP for Beginners · SAP FICO
SAP Programming · SAP HR

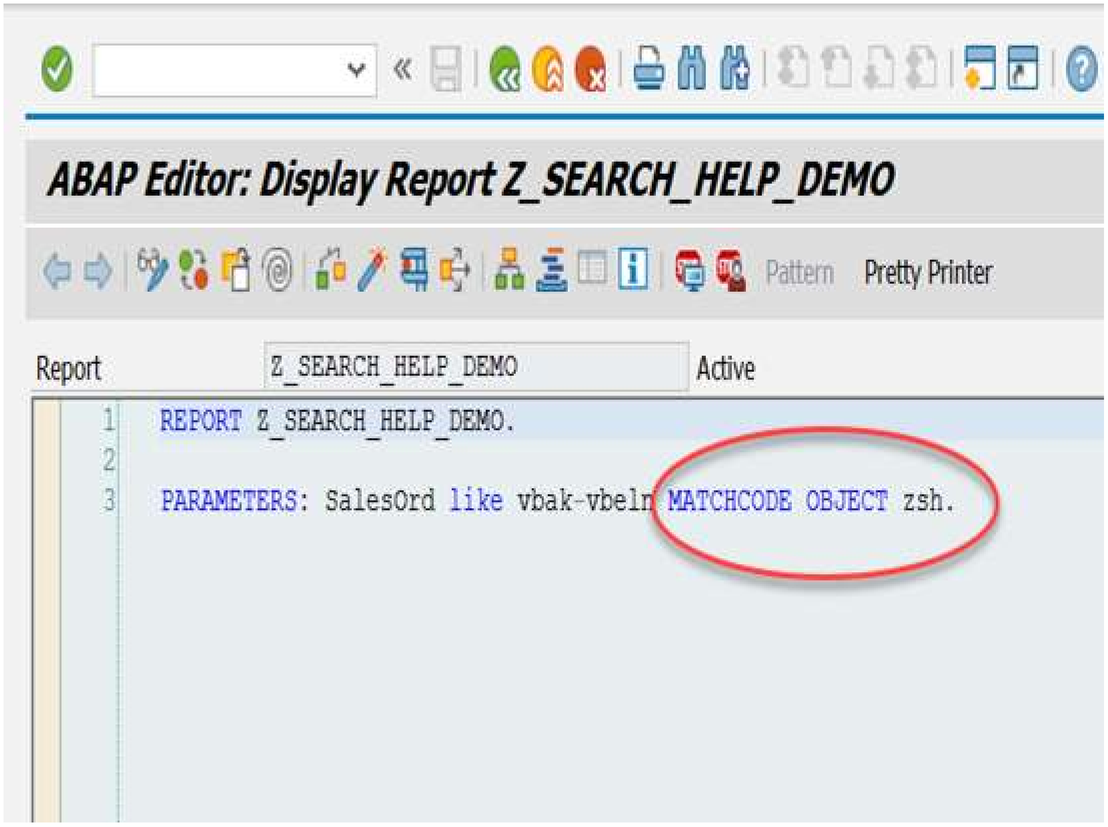
SAP Programming | SAP PRESS Books and E-Books
https://www.sap-press.com/programming/
Learn to code in **ABAP**, SAP's programming language!... **ABAP** Development for SAP HANA.See how SAP HANA has changed **ABAP**!
Advanced ABAP · User Interfaces · Programming E-Bites · ABAP Basics

What if this “type ahead” (Predictive) or “search engine like” functionality could be used in our ABAP 7.4 Search helps?

Well, the good news is it can. You need to be on SAP NetWeaver 7.4 SP06, and SAP GUI 7.30 for Windows Patch Level 5, SAP recommends Patch Level 6 or higher. You can use this on ABAP 7.4 SP05, but you will need to add a PBO section in your DYNPRO to call class CL_DSH_DYNPRO_PROPERTIES=>enable_type_ahead.

At the end of this chapter, I will add a link to a SAP video that explains this and much more in detail. For this blog, I am on the correct support pack and GUI, so this will all be done with any coding changes.

First, let’s look at the code that invokes the Search help. As you can see, this program is a simple one line PARAMETER statement that invokes MATCHCODE OBJECT zsh.

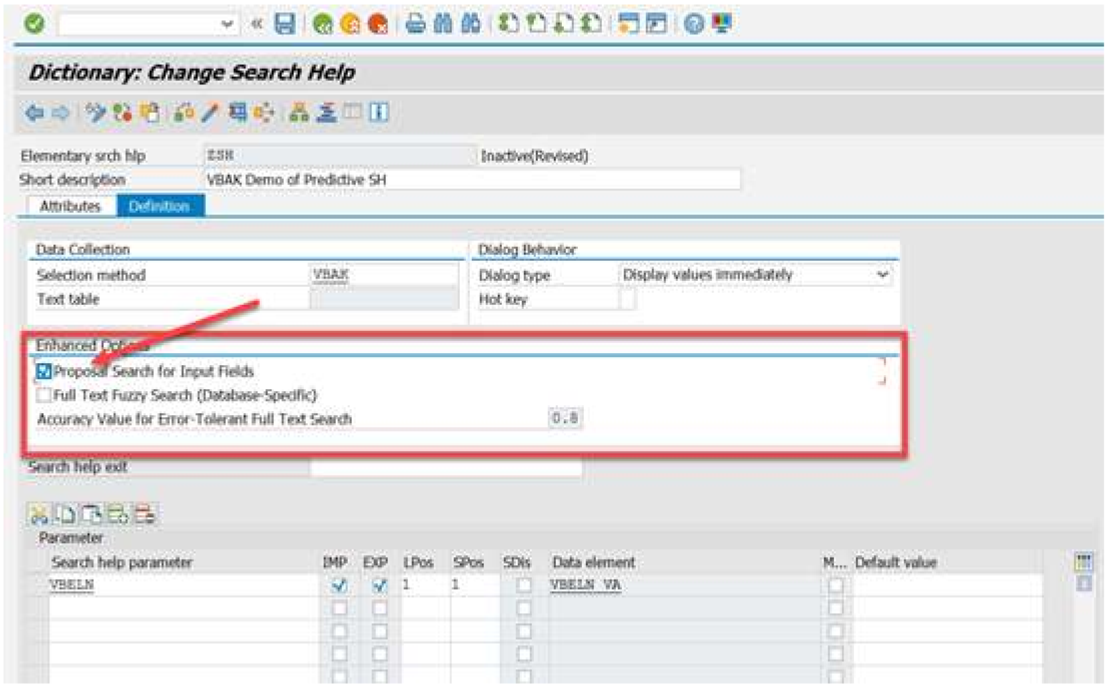


ABAP Editor: Display Report Z_SEARCH_HELP_DEMO

Report Z_SEARCH_HELP_DEMO Active

```
REPORT Z_SEARCH_HELP_DEMO.

PARAMETERS: SalesOrd like vbak-vbeln MATCHCODE OBJECT zsh.
```

Next, let’s double-click on the Search Help zsh in order to enter transaction Se11 and make the changes. Once you are there, please notice a new section called ENHANCED OPTIONS. In this section, you will see a checkbox for “proposal Search for Input Fields”. This is what will allow the type-ahead – aka search-as-you-type – function. Checking this box allows a faster, search engine-like user interaction by showing possible search results from the standard F4 help already in a down box beneath the search field. **This option is Database Agnostic, so you Do NOT need HANA for this.**



Dictionary: Change Search Help

Elementary srch hlp: ZSH — Inactive(Revised)
Short description: VBAK Demo of Predictive SH

Attributes | Definition

Data Collection
Selection method: VBAK
Text table:

Dialog Behavior
Dialog type: Display values immediately
Hot key:

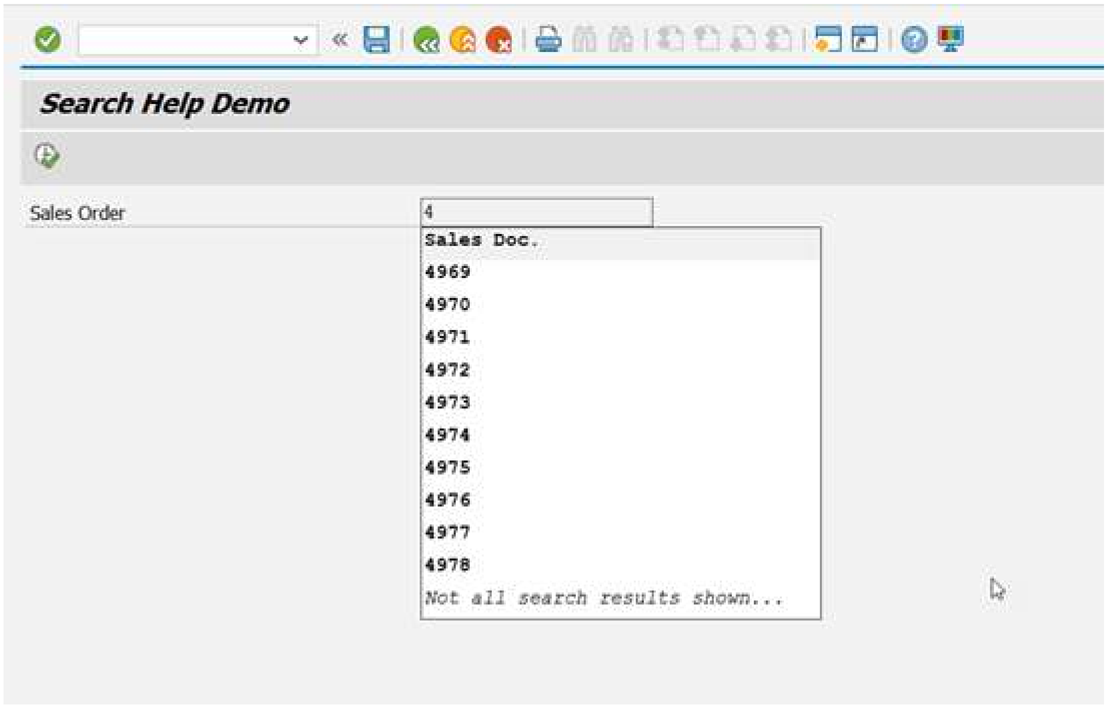
Enhanced Options
[✓] Proposal Search for Input Fields
[ ] Full Text Fuzzy Search (Database-Specific)
Accuracy Value for Error-Tolerant Full Text Search: 0.8

Search help exit:

Parameter

Search help parameter	IMP	EXP	LPos	SPos	SDis	Data element	M...	Default value
VBELN	✓	✓	1	1		VBELN_VA		

OK, now let's activate the changes and test the demo program again. As you see if I begin to enter a number, like 4, I immediately start seeing results.



Search Help Demo

Sales Order: 4

```
Sales Doc.
4969
4970
4971
4972
4973
4974
4975
4976
4977
4978
Not all search results shown...
```

In addition to this “type-ahead” option in the new Enhanced Section of the Search Help, there is the” Fuzzy Search” option. This allows a fault-tolerant, cross-column full-text search. This option doesn’t work on all databases currently – But it does on SAP HANA. An accuracy value can be specified for the error tolerance of the full-text search.

Here is the video I promised.

New Features in ABAP 7.4

Anthony Cecchini is the President and CTO of Information Technology Partners (ITP), an ERP technology consulting company headquartered now in Virginia, with offices in Herndon. ITP offers comprehensive planning, resource allocation, implementation, upgrade, and training assistance to companies. Anthony has over 20 years of experience in SAP business process analysis and SAP systems integration. ITP is an Appian, Pegasystems, and UIPath Low-code and RPA Value Added Service Partner. You can reach him at ajcecchini@itpfed.com.