

# 1818219 - Debugging the payroll Schema

Version 1 from 25.02.2013 in English

## Symptom

Information is required how to debug the payroll schema.

*As a prerequisite to understand this kba you should have a basic debugging knowledge.*

## Reproducing the Issue

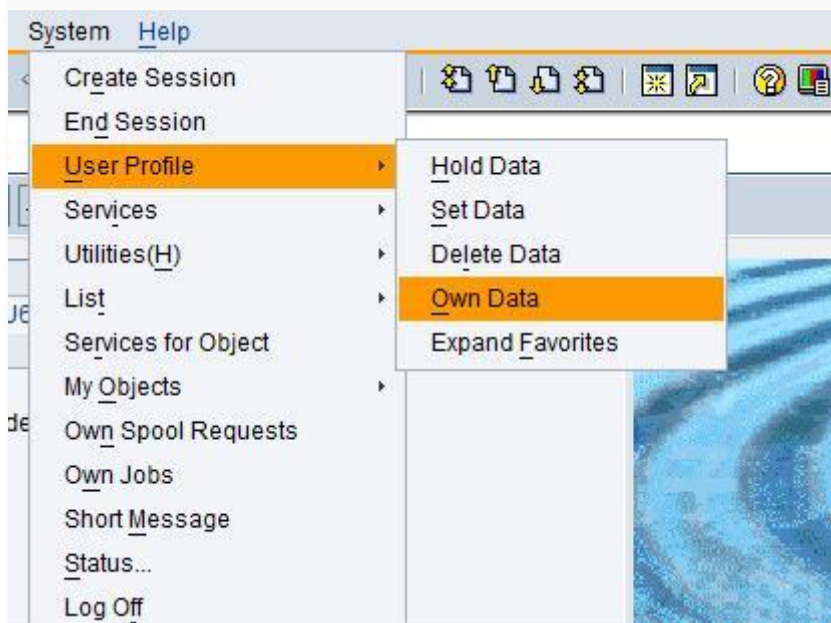
Run Payroll driver RPCALC\*0 / H\*\*CALC0 in order to reproduce the issue.

## Resolution

### Using hard break-points to debug the Schema.

Whenever you can modify the system that you are analyzing (e.g. you are debugging a development system), and you have authorization to modify rules and schemas you can set a hard break-point.

For this, you have to modify your user-profile and set an ABAP prefix (AB4).





User	MARTINEZCA		
Last Changed	MARTINEZCA	23.03.2005	08:37:36

Address

Defaults

**Parameters**

Parameter ID	Parameter value	Short Description
AB4	CME	ABAP prefix

Once you have a prefix of your own, you can set a break-point at a certain point in a Schema by editing the Schema (Trx PE01). For this you'll use the payroll function BREAK followed by your ABAP prefix.

Example: Set a break-point in Schema E000:

000010	COM					Payroll schema: Spain
000020	COM					Program type: PAYROLL
000030	COPY	EIN0				Payroll initialization
000040	COPY	EBD0				Basic data Spain
000050	COPY	ELR0				Reading of last payroll results
000060	BREAK	CME				

If you run your payroll driver, the debugger will stop just before EANT schema.

### Payroll for Spain

Payroll period

Payroll area

E0

☐ Current period
 ☒ Other period
 

1 2008

Selection

Personnel number

35900043

Payroll area

E0

General program control

Pers. calculation schema

E000

Forced retro. acctg from

☒ Test process (no updates)



## ABAP Debugger

ABAP Debugger interface showing the source code of the function module `FORM FUBREAK`. The main program is `RPCALCE0` and the source code of is `RPCMAS09_FUBREAK`.

```
FORM FUBREAK
  FORM fubreak.
  DATA: prefix LIKE sy-prefix.
  CHECK sw_nobreak NE 'X'.
  * GET PARAMETER ID 'AB4' FIELD SY-PREFX.
  GET PARAMETER ID 'AB4' FIELD prefix.
  * IF SY-PREFX EQ AS-PARM1 OR SW_BREAK EQ 'X'.
  IF prefix EQ as-parm1 OR sw_break EQ 'X'.
    BREAK-POINT. "#EC *
  ENDIF.
ENDFORM.
```

Now you can go to the next payroll function by exiting function break (F7).

ABAP Debugger interface showing the source code of the function module `FORM AS-FUNKTION`. The main program is `RPCALCE0` and the source code of is `RPCASF00`.

```
FORM AS-FUNKTION
  FUERPD          "BC ERPD
  FUESVB0         "BD ESVB0
  FUESVC0         "BE ESVC0
  FUEIGA0         "BF EIGA0
  FUEADEV         "C0 EADEV
  FUEPR01         "C1 EPR01
  FUP0480         "C2 P0480
  FUP0092         "C3 P0092
  FUEERE0         "C4 EERE0
  FUEIGA1         "C5 EIGA1
  .
  ENDFORM.
```

Then press key F5 and set a soft break-point at `PERFORM as-funktion`.





Main Program	RPCALCE0
Source code of	RPCHRT09_ASLOOP

FORM ASLOOP

```

                                next_as_entry.
ENDIF.                          "XD0ALRK000847

IF p_with_log NE space.
    PERFORM fill_log_keys USING 'A'    "!"
                                asnum  "!"
                                aper_numb
                                next_as_entry.
ENDIF.
PERFORM refresh_function_text_table.
 PERFORM as-funktion.
 DESCRIBE TABLE ptext LINES ptext_lines.

```

You can display the current function to be processed by displaying in your debugger screen table AS (Header).

Table AS contains the Schema and its header contains the current function in the schema being processed (AS-FUNCO).

Structured field		as		
Initial Length (in Bytes)		50		
No.	Component name	Ty...	Lngh	Contents
1	FUNCO	C	5	BREAK
2	FNUMB	X	1	06
3	PARM1	C	4	CME
4	PARM2	C	4	
5	PARM3	C	4	
6	PARM4	C	4	
7	PROTO	C	1	
8	SKIP	P	3	1



Debugging Tratar Pasar a Breakpoints Opciones Desarrollo Sistema Ayuda

Modo debugging

Campos Tabla Breakpoints Watchpoints Llamadas Resumen Opciones

Programa ctrl. RPCALCE0

Cód.fuente de RPCHRT09\_ASLOOP

FORM ASLOOP

next\_as\_entry. "!"

ENDIF. "XD0ALRK000847

Tabla interna ps Tipo STANDARD Formato E

1	FUNCO	FNUMB	PARM1	PARM2	PARM3	PARM4	PROTO	SKIP
	P0092	C3						1
1	BLOCK	73	BEG					1
2	BLOCK	73	END					1
3	BLOCK	73	BEG					1
4	ENAME	16						1
5	WPBP	60						1
6	P0061	B4						1
7	P0480	C2						1
8	P0062	B5						1
9	P0092	C3						1
10	GON	1C						1
11	BLOCK	73	END					1
12	BLOCK	73	BEG					1
13	IMPRT	22	L					1
14	PORT	4A	E006	P06	NOAB			1
15	SETCU	57						1
16	BLOCK	73	END					1
17	BLOCK	73	BEG					1
18	P0092	C3						1
19	EADEV	C0						1
20	BLOCK	73	END					1

Now you can stop right before next function to be processed in schema by pressing F8.



FORM ASLOOP

```

ENDIF.
PERFORM refresh_function_text_table.
PERFORM as-funktion.

```

"XFG note 607609  
"XD0ALRK000847

Structured field

Initial Length (in Bytes)

No.	Component name	Ty...	Lngh	Contents
1	FUNC0	C	5	P0092
2	FNUMB	X	1	C3
3	PARM1	C	4	
4	PARM2	C	4	
5	PARM3	C	4	
6	PARM4	C	4	
7	PROTO	C	1	
8	SKIP	P	3	1

In this case, you are about to debug function P0092. To get inside it just press F5.

FORM FUP0092

```

endprovide.
perform first-st.
first-st-begda = st-begda.
endform.      "END OF FUP0062

*-----
* Cálculo de Antigüedad
*-----

form fup0092.
* Lectura del Infotipo P9092
provide * from p0092
    between pn-begda and pn-endda.
endprovide.
endform.      "END OF FUP0092

```

In this case the function code to be debugged is very short. Go to the next function by pressing F8.

Of course you could have set a soft break-point directly at function P0092--- pe04. You can set a break-point in a payroll rule, as well. For this you'll use the payroll operation BREAK, followed by your ABAP prefix.

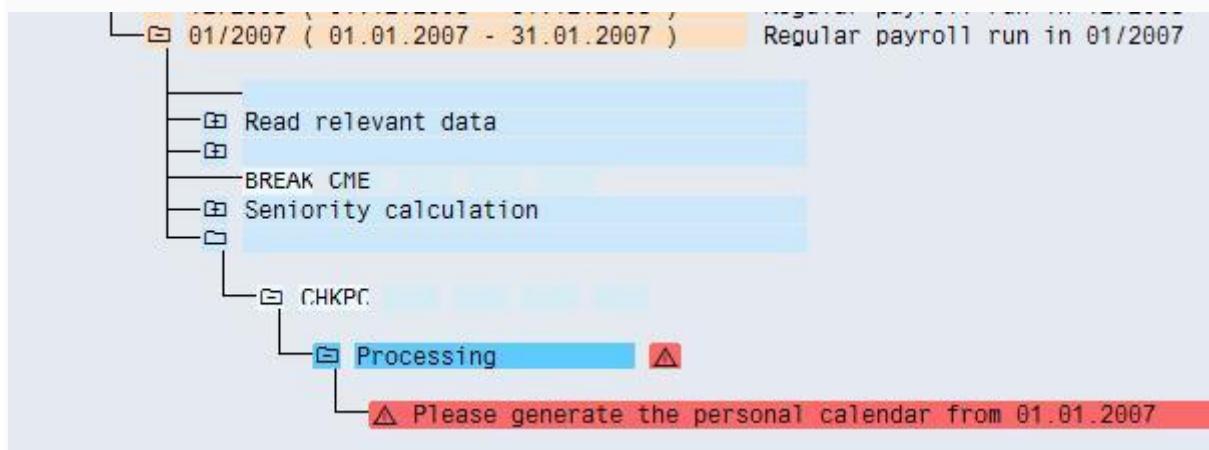


## Edit Rule: ZOSO ES Grouping 3 Wage Type/Time Type M101

Cmmnd		Stack								
Line	Var.Key	CL	T	Operation	Operation	Operation	Operation	Operation	Operation	*
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----										
000010				BREAK CME						

### When a pernr is rejected in the payroll log.

Just set a break-point in the command "reject". Start payroll processing in debugging mode /H, then go to... finally press F8 key, you'll reach the reject command. After that you have to take a look at "Calls" to guess where the error is coming from. For example, you run the payroll drive and get a log like this:



Go back to the selection-screen and start payroll in debugging mode with /H.



Program Edit Goto System Help

/H

## Payroll for Spain

Selections from Search helps

Payroll period

Payroll area E0

☐ Current period

☒ Other period 1 2009

Selection

Personnel number 35900006

Payroll area

General program control

Pers. calculation schema E000

Forced retro. acctg from

☐ Test process (no updates)

Now you are debugging the payroll and you need to reach the point of rejection:

Debugging Edit Goto Breakpoints Settings Development System Help

ABAP Debugger

Save Ctrl+S

Breakpoint at Statement... Shift+F5

Create/delete Shift+F4

Delete all Shift+F2

Deactivate/Activate

Deactivate all

Activate all

Create watchpoint Shift+F8

Subroutine... Shift+F6

Function module.... Shift+F7

Method...

Exception...

System exception

Fields Table

Main Program RPCALCE

Source code of RPCHRTG

FORM %\_SEL\_SCREEN\_SCI

```

IF sy-subrc <> 0.
    SELECT SINGLE * FROM t500t WHERE molga = calcmolga
    AND spras = sy-langu.
    MESSAGE ID '3G' TYPE 'E' NUMBER '800' WITH payty t500t-ltext.
ELSEIF payty = cd_c-supplemental.

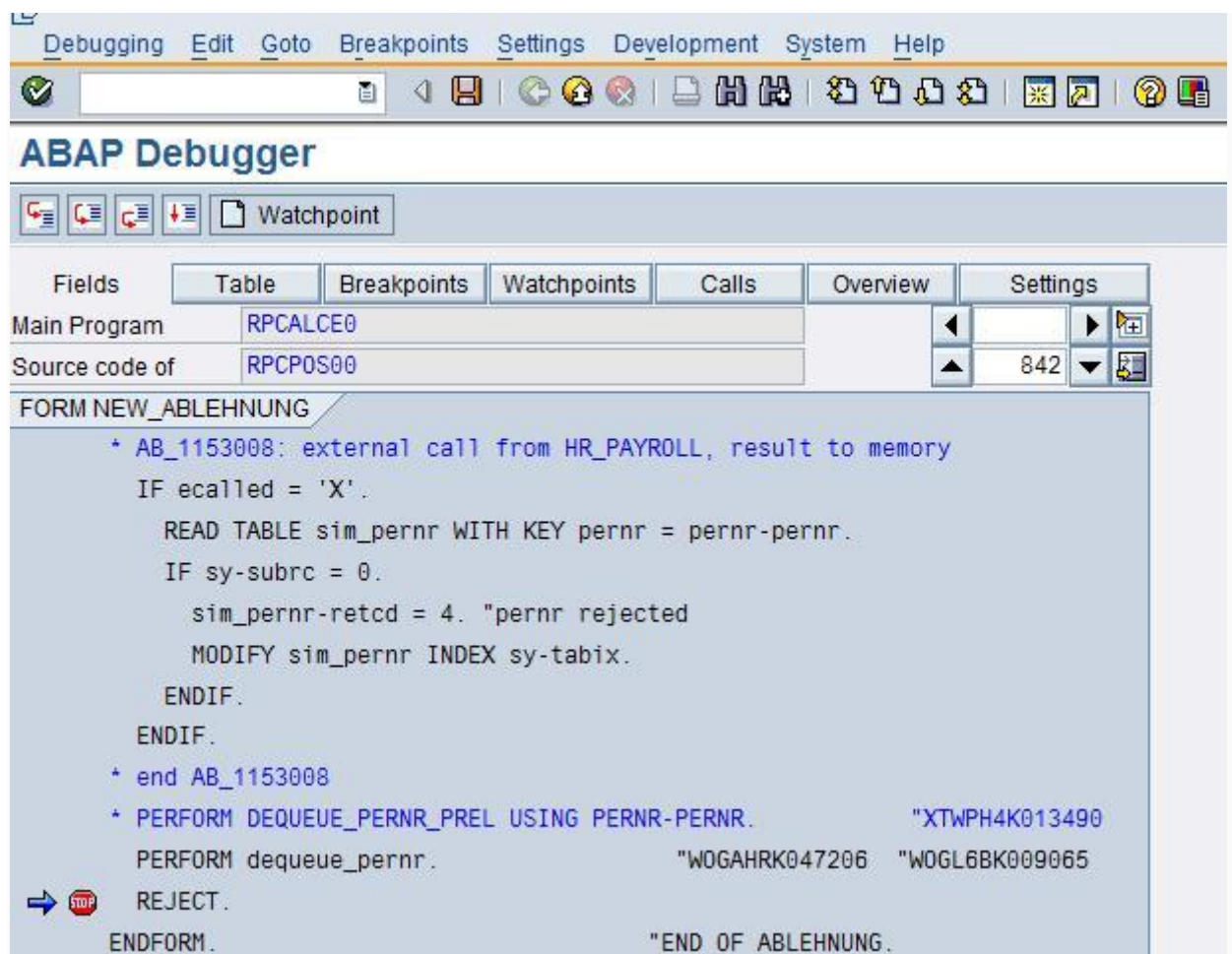
```

Stop it at the reject command:









Now press the "Calls" icon:



#### FORM NEW\_ABLEHNUNG

```
* AB_1153008: external call from HR_PAYROLL, result to memory  
IF ecalled = 'X'.  
    READ TABLE sim_pernr WITH KEY pernr = pernr-pernr.
```

#### Act call stack

	No.	Program	Type	Processing block
➡	17	RPCALCE0	FORM	NEW_ABLEHNUNG
	16	RPCALCE0	FORM	FILL_MSGTAB_FINAL_STEP
	15	RPCALCE0	FORM	FILL_MSGTAB
	14	RPCALCE0	FORM	ERRORS
	13	RPCALCE0	FORM	FUECHKPC
	12	RPCALCE0	FORM	AS-FUNKTION
	11	RPCALCE0	FORM	ASLOOP
	10	RPCALCE0	FORM	MONATSABRECHNUNG
	9	RPCALCE0	FORM	RUECKRECHNUNG
	8	RPCALCE0	FORM	MAIN
	7	RPCALCE0	FORM	%_GET_PERNR
	6	SAPDBPNP	FORM	FILL_INFOTYPE_TABLES_AND_PUT
	5	SAPDBPNP	FORM	PUTPERN
	4	SAPDBPNP	FORM	LOOP_AT_INDEX_AND_PUT
	3	SAPDBPNP	FORM	PUT_PERNR
	2	SAPDBPNP	FORM	%_ROOT
	1	RPCALCE0	EVENT	SYSTEM-EXIT

The following routines:

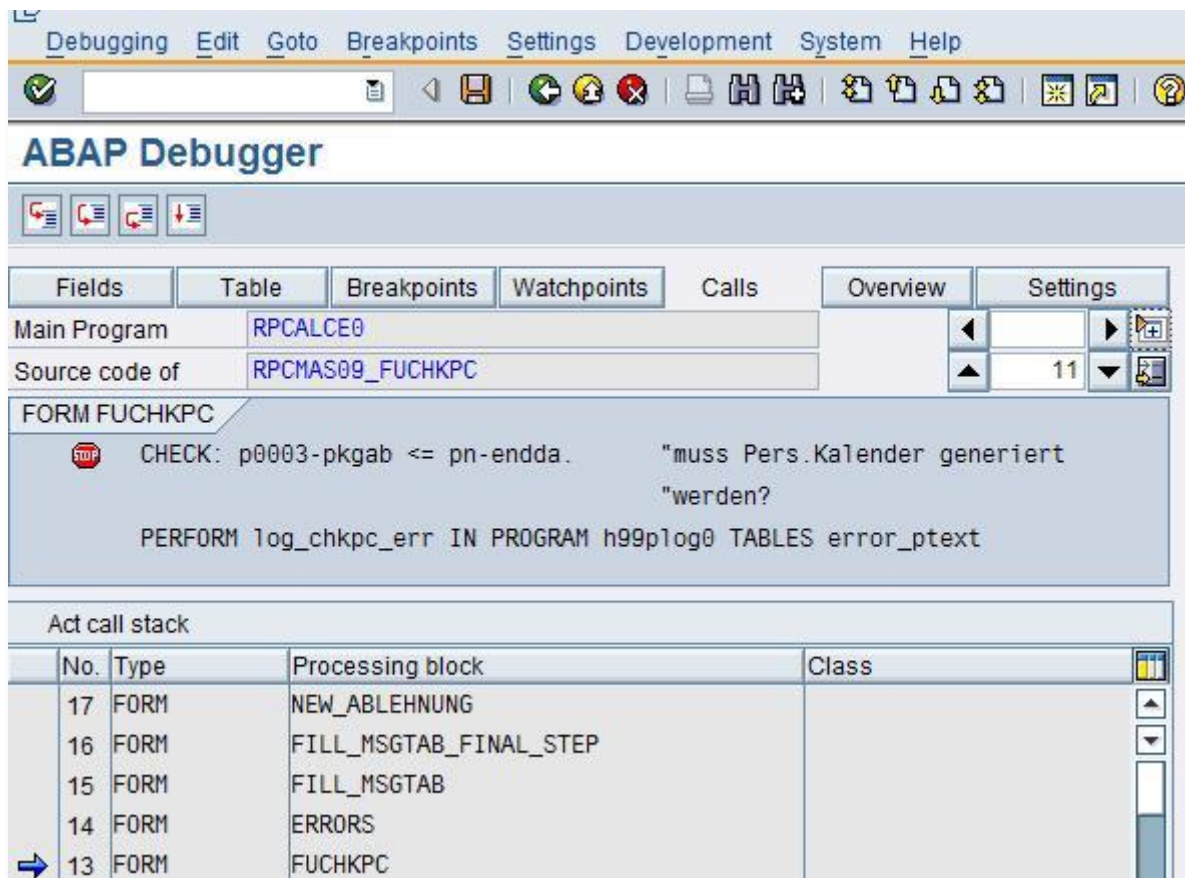
FORM NEW\_ABLEHNUNG  
FORM FILL\_MSGTAB\_FINAL\_STEP  
FORM FILL\_MSGTAB  
FORM ERRORS

are common to all the rejections so they will not add any value info, so we go to the first relevant form:

FUECHKPC

Double-clicking on it we get to the ABAP code and set a break-point just before the error process is triggered (Or maybe analyzing the abap code that triggers the error is enough):





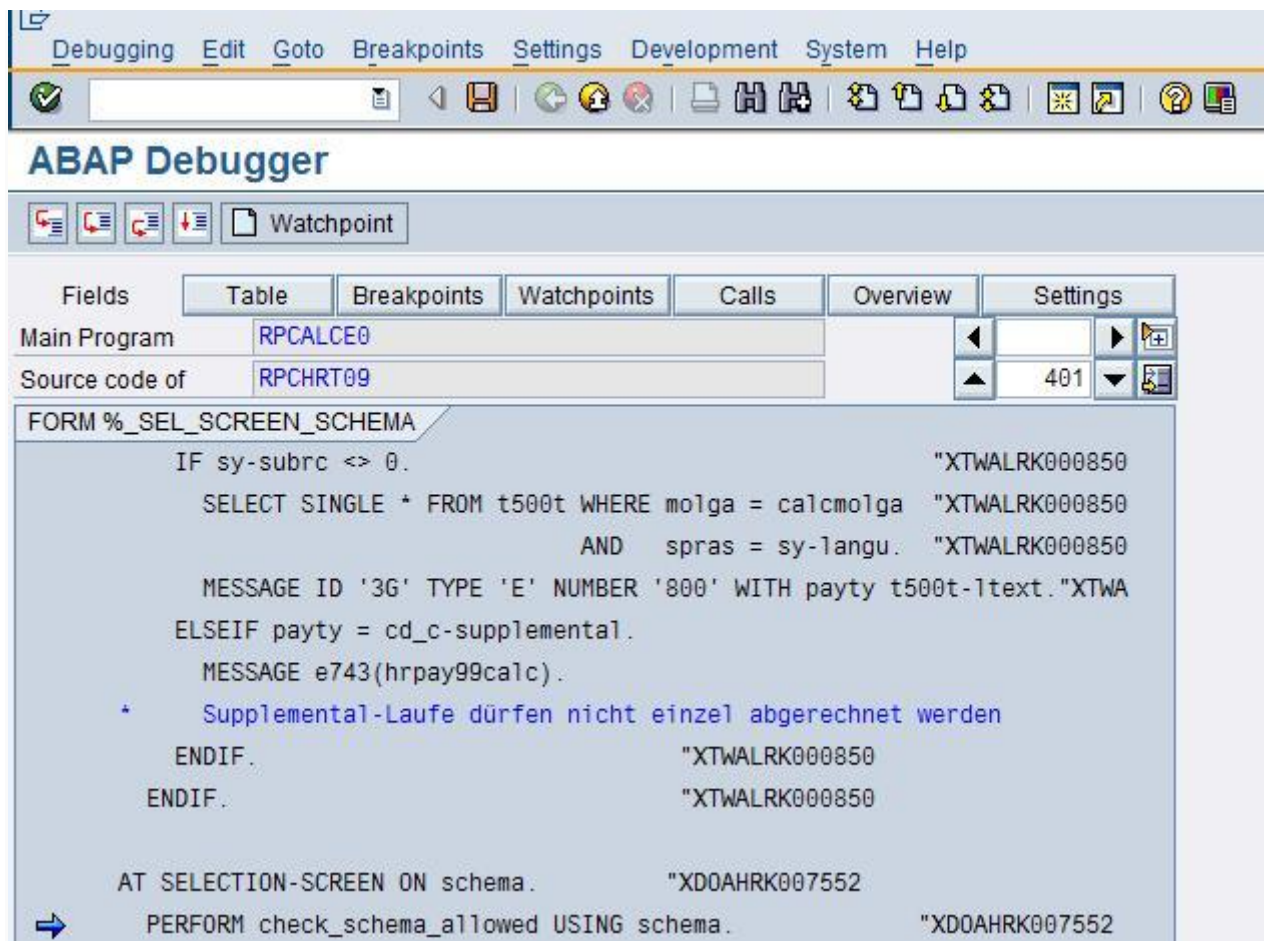
When you start payroll again you'll be able to debug the code that triggered the rejection which will help you identifying the cause of the error.

**Find out at which point of the payroll Schema, a Wage Type is generated or a WT is set to a specific value.**

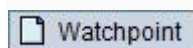
Watch-points will help you a lot on this. If for example you find that WT /341 equals 1.375,00 EUR at the final payroll results and you want to know where in the payroll schema WT /341 amount was set to 1.375,00 EUR.

Start payroll in debugging mode by going to the payroll driver selection-screen and setting /H at the ok-code. Then press F8 to start the program.

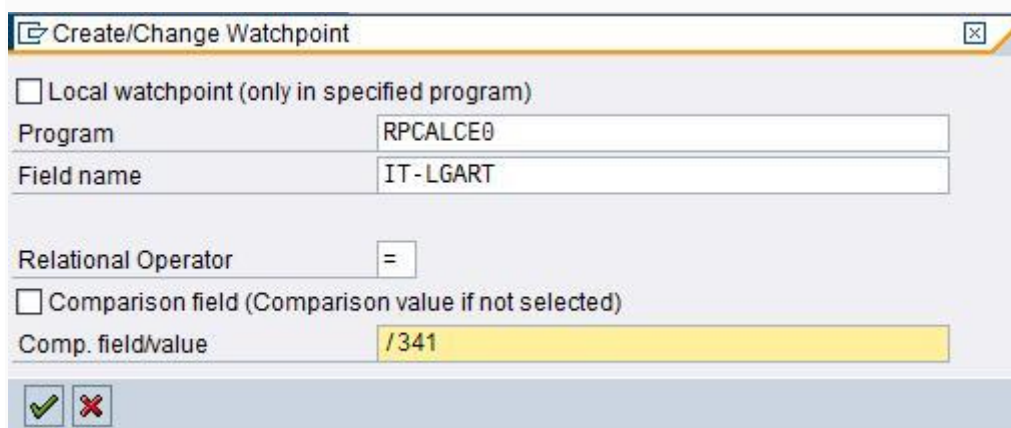




Now create a watchpoint.



And fill the required condition.



Create a second watchpoint because the condition is: it-lgart = /341 and it-betrg = 1.375,00



**Create/Change Watchpoint**

☐ Local watchpoint (only in specified program)

Program:

Field name:

Relational Operator:

☐ Comparison field (Comparison value if not selected)

Comp. field/value:

☐ ☐

Now both condition have to be linked by the "And" operator.

**Watchpoints**

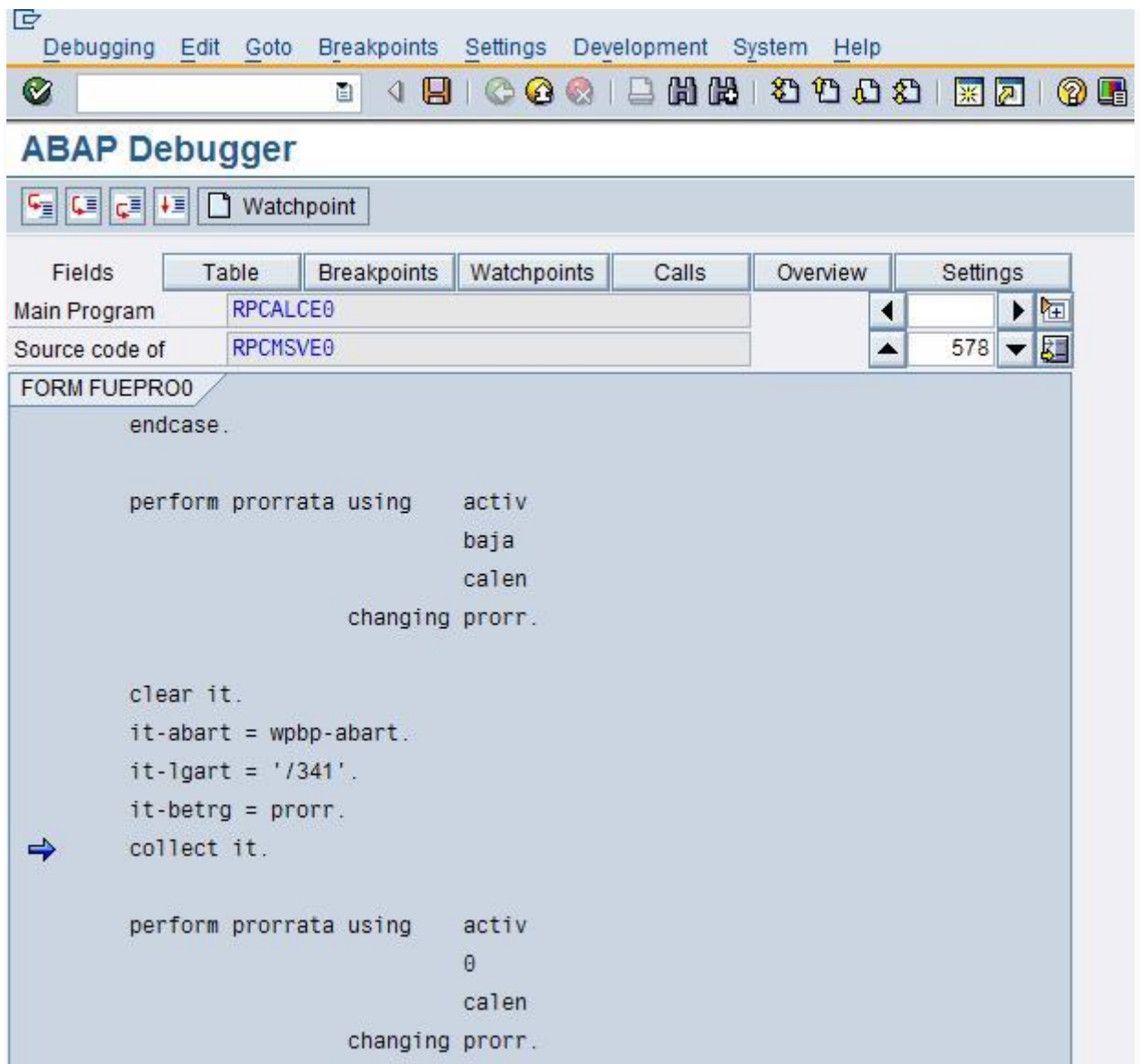
No.	Lo...	Program	Field name	O...	Fld	Comp. field...	
1	<input type="checkbox"/>	RPCALCE0	IT - LGART	=	<input type="checkbox"/>	/341	
2	<input type="checkbox"/>	RPCALCE0	IT - BETRG	=	<input type="checkbox"/>	14	
3	<input type="checkbox"/>						
4	<input type="checkbox"/>						
5	<input type="checkbox"/>						

Logical operator between watchpoints: ☐ OR ☒ AND

Current field contents of the last watchpoint reached:

Now continue by pressing F8. The program will stop when the watchpoint condition is fulfilled.





You'll notice that in the watchpoint condition I used WT table IT instead of table RT. If I had set the same condition for table RT, the program would have stopped later on in the schema when WT /341 is collected to be stored in table RT which is useless. Sometimes, WTs are directly created in table RT so in that cases, setting the condition for table RT might be useful. Many times, WTs are created within a rule operated by function PIT or PRT. In that cases, you'd better set the condition for table OT (OT-LGART, OT-BETRG...). If you do so, the program will be stopped at the relevant operation within the rule. To see which rule is it, look internal table AS header. To know what operation is it, display the structured field OP.



Fields	Table	Breakpoints	Watchpoints	Calls	Overview	Settings
Main Program	RPCALCE0					◀ ▶
Source code of	RPCBU209					▲ 296 ▼
<div>FORM VCOLLECT</div> <pre> ot-lgart = op-lgart. ENDIF.                                "SY-SUBRC EQ 0. ELSE. </pre>						
Structured field		bp				
Initial Length (in Bytes)		20				
No.	Component name	Ty...	Lngh	Contents		
1	OPCOD	C	5	ADDWT	▲ ▼	
2	MODIF	C	1			
3	LGART	C	4	/398		

By the way, if a WT is set to a specific value in a PIT rule and instead of setting the watchpoint at OT table you do it at IT table, the watchpoint will stop at the end of the rule as you can see in the code below:

```

FORM fupit.
LOOP AT it.
plog4_perform plog_header_cycle(h99plog0)
using it-lgart calcmolga.
MOVE ccycl TO i52c5.
MOVE-CORRESPONDING it TO i52c5.
ot = it.
PERFORM regel.
plog1_perform plog_check_rule_performed(h99plog0).
ENDLOOP.
REFRESH it.
PERFORM ot-in-it-append. "append statt collect, beinhaltet refresh ot.
ENDFORM. "END OF FUPIT

```

So it's more accurate to set the watchpoint at table OT than at table IT in these cases, but remember that rules are flexible and there are operations to append WTs directly to table IT.

### Problems with Retros.

When there are many retro periods in the payroll run you are analyzing, the same piece of code will be reached once and again and you have to press the F8 key a lot of times before you reach the point you where the issue arises. Table APER manages the periods processed in a payroll run and it's header holds the currently processed period. In the following pic you can see that period 01.2008 is being processed in a payroll run started from in-period 01.2009.




Field names	1 - 4	Field contents
APER-PAPER	200801	
APER-IAPER	200901	
SY-SUBRC	0	SY-TABIX 23 SY-DBCNT 1

You can use APER-PAPER and APER-IAPER as watchpoint conditions so if for example you set a break-point at function EST00 (Which calculates Spanish taxes) and there are 10 retro periods to be calculated and you want to debug just the current period:

1. Deactivate (Not delete) the break-point at function EST00.
2. Set APER-PAPER = APER-IAPER as a watchpoint condition.

As an alternative to using table APER, you can assign a counter to the break-point so that it stops only after "n" occurrences:

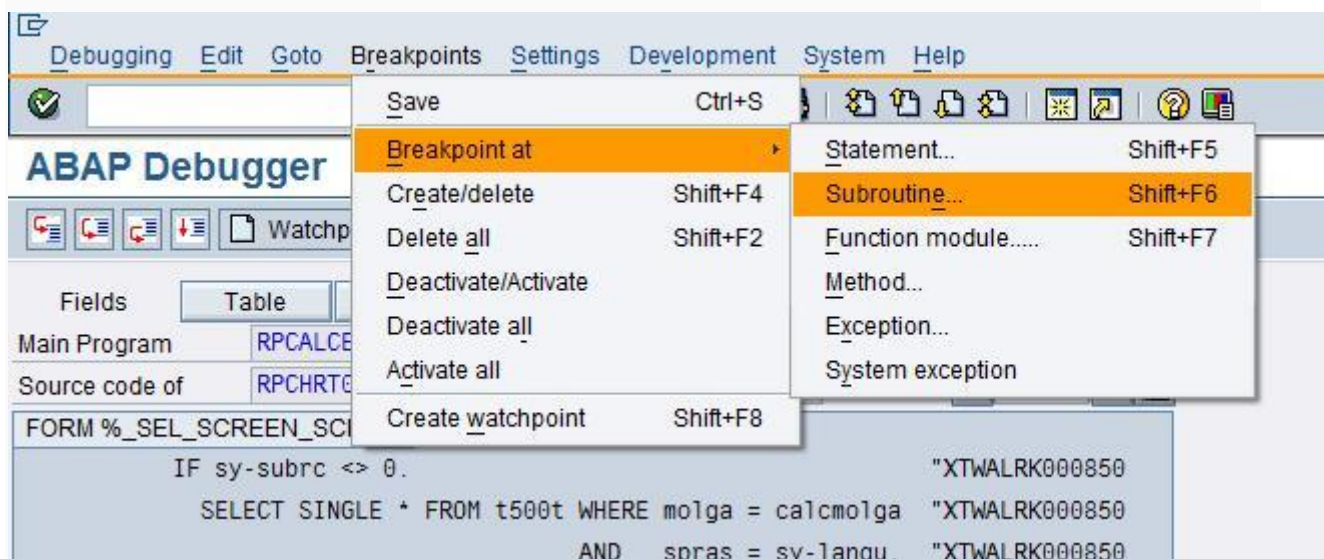
e.g. Stop in function EST00 after EST00 being processed 13 times:

Breakpoints			
	No.	in (absolute path)	Count
	1	FUEST00	PROGRAM=RPCALCE0 13

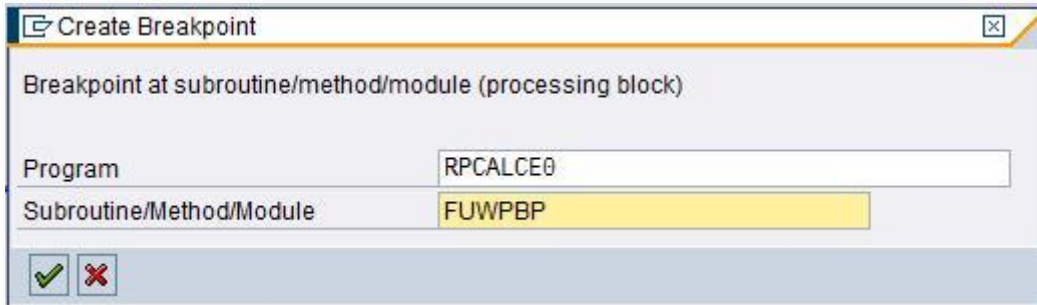
Finally, when the APER-PAPER is set to the payroll period you want to analyze (because the watchpoint stopped), activate the break-point and go ahead: F8.

### Reaching a payroll function or operation once the debugger is started:

Payroll functions are coded as form FUXXXX being XXXX the name of the function. e.g. function wpbp is coded as form fuwpbp. Payroll Operations are coded as OPXXX being XXXX the name of the operation. Knowing this makes easy to reach any Function / Operation.







Breakpoint at subroutine/method/module (processing block)

Program:

Subroutine/Method/Module:

☒ ☐

Then press F8 key and you get to the beginning of the routine.

### How to debug a Personnel Calculation Rule:

You already know that a PCR or cycle is not working as desired but do not know at what point of the rule the problem is. When a PCR is processed, a sequence of payroll operations will be processed. You can see how a rule works processing one operation in each step and monitor the intermediate results. Then you may realize the problem is that the rule is not correct or you can debug inside the operation.

Firstly you have to reach the rule you are going to debug. For that you can set a hard break-point as stated at the beginning of this document or a watchpoint (AS-PARM1 = XXXX where XXXX is the name of the rule) AND ( OT-LGART = YYYY where YYYY is the WT to be processed by the PCR). It is important that both watchpoint conditions are linked by the condition AND

Example:

We are going to debug PCR ESPB at Payroll Schema E000 for payroll driver RPCALCE0.

Going to the payroll log we see that WT S121 is calculated at PCR ESPB:

12/2006 ( 01.12.2006 - 31.12.2006 )		Regular payroll run in 12/2006	
<input checked="" type="checkbox"/>	Read relevant data		
<input checked="" type="checkbox"/>	Read last payroll results		
<input checked="" type="checkbox"/>	Seniority calculation		
<input checked="" type="checkbox"/>	Time data process		
<input checked="" type="checkbox"/>	Read additional payts/deds(10014,0015)		
<input checked="" type="checkbox"/>	Travel expenses		
<input checked="" type="checkbox"/>	Special payments calculation		
<input checked="" type="checkbox"/>	IMPRT L		Import last result
<input checked="" type="checkbox"/>	DATES		Complete dates data table
<input checked="" type="checkbox"/>	SPA 2		SPECIAL PAYTS (calc. entitlement part)
<input checked="" type="checkbox"/>	PIT ESPB		Create valuation bases
<input checked="" type="checkbox"/>	IMPRT L		Import last result (EMFI)
<input checked="" type="checkbox"/>	SPU00 0000		Special payment transfer period
<input checked="" type="checkbox"/>	SPC 2		SPECIAL PAYMENTS (valuation and gener.)
<input checked="" type="checkbox"/>	PIT XSPD P48 NOAB		Delete split CNTR3
<input checked="" type="checkbox"/>	EPR00 ESPB 1		Calculation of Social Insurance prorate



PIT ESPB Create valuation bases

Input

Table IT

A	Wage type	APC1C2C3ABKoReBTAvvTvn	One amount/one number	Amount
3	/001 Valuation b01		27,55	
3	/001 Daily basis01		154,02	
3	AN01 Seniority: 01		3.183,00	4,00 3.183,00
3	ANTI Seniority 01			3.183,00
3	M101 Basic stand01			1.391,49
3	M102 Standard bo01			200,00
3	/3CN Days month		31,00	
3	/3CA Monthly day		30,00	
3	/3AN Active days		31,00	
3	/3AC Active days		30,00	
3	S121 Basic stand01	01	1,00	
3	S122 Standard bo01	01	1,00	

Processing



S122 Standard bonus SP 2

Rule	ESGPCR	VaKey	Operation
ESPB	3		RTE=1.00
ESPB	3		ADDWT&FAC1
ESPB	3		GCYGESPB2
ESP2	3		ELIMI *
ESP2	3		RESET AR
ESP2	3		AMT= M102
ESP2	3		AMT+ INCE
ESP2	3		RESET AR3
ESP2	3		STAB SP
ESP2	3		RTE=BBSGRD
ESP2	3		MULTI RAR
ESP2	3		RTE/100
ESP2	3		ZERO= A
ESP2	3		RTE/& FAC1
ESP2	3		ADDWT *
ESP2	3		ZERO=&FAC1

Output

Table IT

A	Wage type	APC1C2C3ABKoReBTAvTvn	One amount/one number	Amount
3	/001 Valuation b01		27,55	
3	/00I Daily basis01		154,02	
3	/3AC Active days		30,00	
3	/3AN Active days		31,00	
3	/3CA Monthly day		30,00	
3	/3CN Days month		31,00	
3	AN01 Seniority: 01		3.183,00	4,00
3	ANTI Seniority 01			3.183,00
3	M101 Basic stand01			1.391,49
3	M102 Standard bo01			200,00
3	S121 Basic stand01	01	1.391,49	
3	S122 Standard bo01	01	200,00	

So now, let's start the debugger and set the watchpoint condition AS-PARM1 = ESPB and OT-LGART = S121, here is how your debugger screen will look when the watchpoint conditions are reached:



## ABAP Debugger

The screenshot shows the ABAP Debugger interface. At the top, there is a menu bar with icons for 'Watchpoint' and other debugging functions. Below this is a tabbed view with tabs for 'Fields', 'Table', 'Breakpoints', 'Watchpoints', 'Calls', 'Overview', and 'Settings'. The 'Fields' tab is active, showing 'Main Program' as 'RPCALCE0' and 'Source code of' as 'RPCMAS09\_FUPIT'. The main code area displays the source code of the 'FORM FUPIT'. The code is in German and includes comments and statements. A blue arrow points to the 'PERFORM regel.' statement, which is the target of the next step in the tutorial.

```

* IT EINTRAG FUER EINTRAG NACH DEN HIERFUER IN DER TABELLE I52C5
* VORGESEHENEN REGELN. DURCH DIE REGELN WIRD DIE BEARBEITUNG IM
* EINZELNEN VERANLASST. DABEI ENTSTEHT EINE AUSGABETABELLE OT, DIE
* DASSELBE FORMAT WIE DIE EINGABETABELLE HAT. SIE WIRD ZUR EINGABE-
* TABELLE DER NAECHSTEN PHASE.
* -----
ccycl = as-parm1.
* perform phase-heading. "XDOALRK030399
LOOP AT it.
  plog4_perform plog_header_cycle(h99plog0)
    using it-lgart calcmolga. "XDOALRK000847
  MOVE ccycl TO i52c5.
  MOVE-CORRESPONDING it TO i52c5.
  ot = it.
  PERFORM regel.
  plog1_perform plog_check_rule_performed(h99plog0). "XDOALRK000847
ENDLOOP.
REFRESH it.
PERFORM ot-in-it-append. "append statt collect, beinhaltet refresh ot.
ENDFORM. "END OF FUPIT

```

Rule ESPB is processed by PIT Function which means that table WTs in table IT will be processed in a loop. Each time a WT is processed the table IT header will be copied to auxiliary table OT and within the PCR processing the individual operations will normally put their results in table OT. At the end of the PCR processing, table OT will substitute table IT. Relevant fields for IT and OT are ( LGART, BETRG, BETPE, ANZHL...).

Now go inside PERFORM regel by pressing F8 and search PERFORM boper with the arrow down

-> Set a break-point on PERFORM boper.



```

FORM REGE1
*      MOVE t52c5+3 TO i52c5.                                "W0GL6BK009065
      i52c5 = t52c5_wa-rules.                                "!"
*      perform pr-rege1.                                     "XDOALRK030399
ELSE.
      PERFORM rege1_error(h99plog0) TABLES error_ptext
                                USING 'R21'
                                i52c5(18).
      PERFORM errors TABLES error_ptext.
ENDIF.
ENDIF.
ENDIF.
ENDIF.
IF nextr NE 'N'.
  WHILE nextr NE 'N'.
    ↳ lastrule = i52c5.
      nextr = 'N'.
      PERFORM boper.                                "Operationen einer Regel abarbeiten.

```

Make sure you display at the bottom of the debugger screen the fields you want to monitor:




Field names	1 - 4	Field contents
ot-1gart		S121
ot-betpe		1.00
ot-betrg		0.00
ot-anzh1		0.00
SY-SUBRC	0	SY-TABIX 0 SY-DBCNT 25 SY-DYNNI

You can also display the current operation at the header of table OP or i52c5-op1.

Now you can deactivate the watchpoint.

Click in F8 every time you want to go to the next operation (And F5 if you want to take a deeper look as to how the operation works internally):

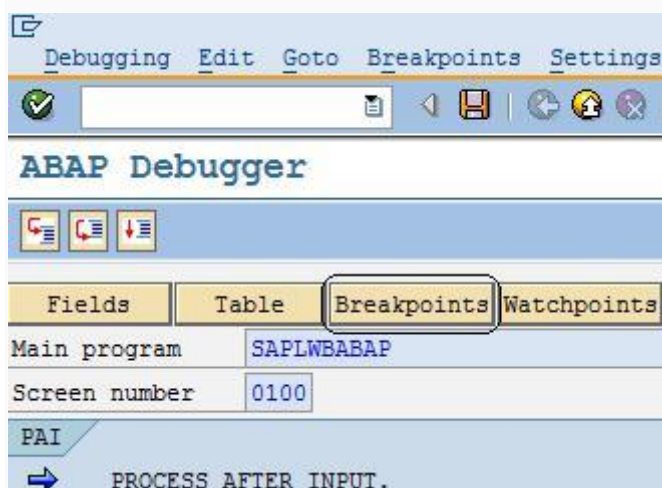


h52c5-op1	RTE=1.00
ot-lgart	S121
ot-betrg	0.00
ot-betpe	1.00
	
i52c5-op1	ELIMI *
ot-lgart	S121
ot-betrg	0.00
ot-betpe	1.00
	
h52c5-op1	RTE=BBSGRD
ot-lgart	S121
ot-betrg	1391.49
ot-betpe	1.00
	
h52c5-op1	ZERO=&FAC1
ot-lgart	S121
ot-betrg	0.00
ot-betpe	1391.49

### Using counters:

It might happen that you set a breakpoint at a certain part of the code where the payroll execution goes through again and again. For example in retrocalculation or in case a your break-point is in a routine that is placed inside a loop. E.g. you placed a break-point inside a piece of code that is executed within a loop and you want the debugger to stop only the 10th time that the break-point is reached. You can specify a counter for your break-point.

Click on the break-point button:





Set 10 times in the count. field.


FUNCTION S\_UI\_CLASS\_DEPENDENCIES

<fs\_ui\_flag>

TYPE flag.

DATA:

Breakpoints

No.	in (absolute path)	Coun...
 1	LURL_GENERATIONU23(19)	10

Of course, if the break-point is reached less than 10 times, it will never stop, but you are supposed to know the number of times the break-point will be reached before hand.

### When an issue happens for a specific pernr

The issue only happens with a specific pernr X, however if you run payroll driver for the pernr X individually, you don't get the error.

1. (Optional) You can set the break-point beforehand in the piece of code you want to debug.
2. (Conditional) If you didn't do step (1) start payroll driver with /H.
3. (Conditional) If you did step (1) deactivate all break-points.
4. Set a watch-point with condition pernr-pernr = X.
5. (Conditional). If you did step (1) Activate all break-points, otherwise set a break point in the piece of code you want to debug.

### Can all this be applied to the Time Evaluation?.

Yes, RPTIME00 works with the rules in the same way as the payroll drivers RPCALCX0 / HXXCALC0. In this case you may find useful to know that PER play the same role as table APER for payroll. Therefore you can use the header of PER to stop the processing at a specific day:

e.g. Set a watchpoint at PER-BEGDA = 20080901 or ACDATE = 20080901.

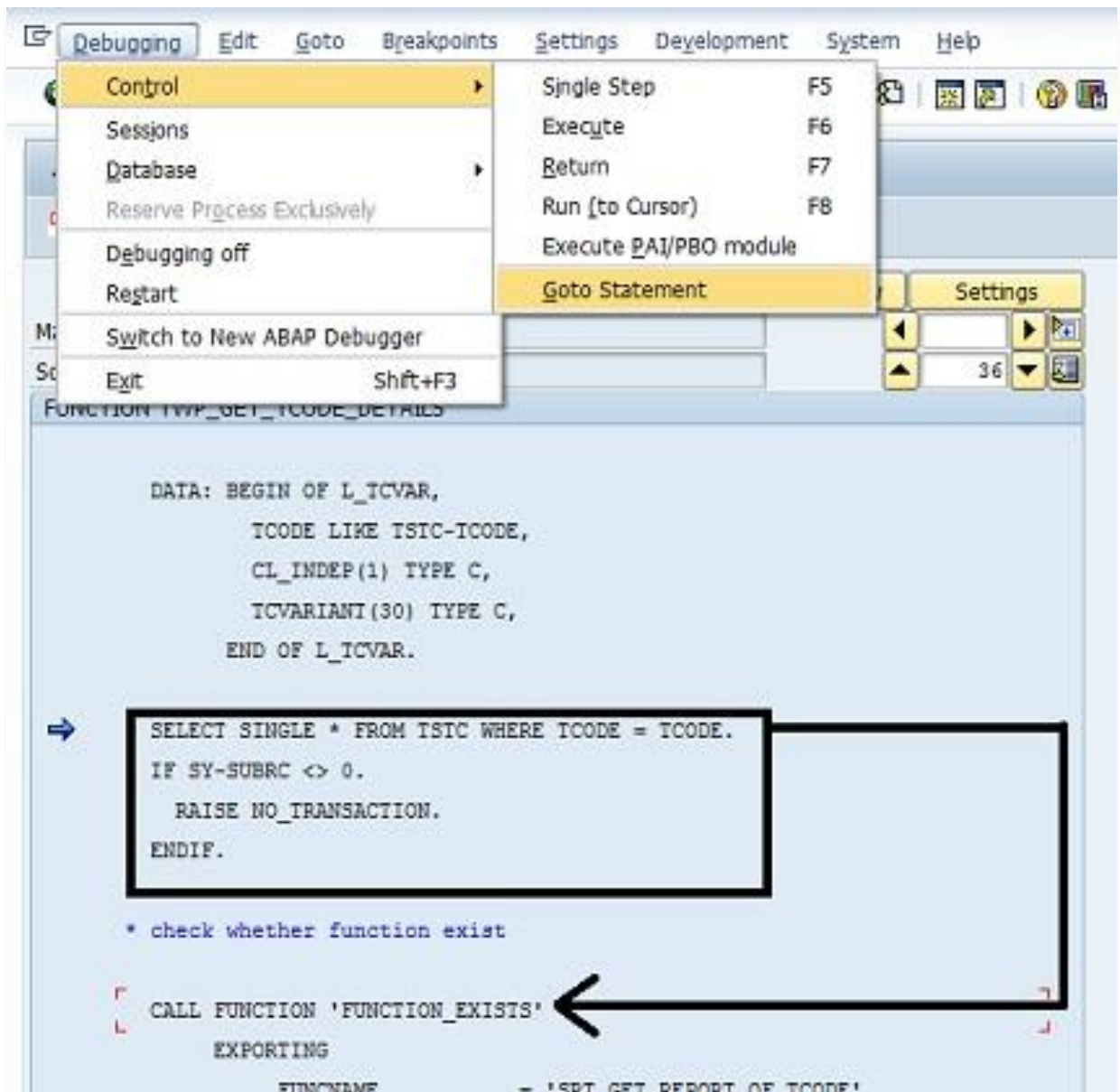
### Good to know:

You can expand the payroll an time schema by running program RPDASC00.

### Skipping blocks of code:

As you can see in the picture below, in new releases you can skip a piece of code by placing the cursor in the code line you want to jump:





This is not a feature valid only for payroll or HCM module. Skipping a block code is useful:

\* If you have an error and you suspect which the code responsible for the error is. Just skip it and you'll know if your suspicion is true or not. If the code was introduced by an SAP note, you may have found a side-effect. And it's not good to modify a standard program just to do this kind of tests.

\* After a long debug session you executed a part of the code that was important without pay attention. In this case you may jump back to a code line that was already executed as long as you do not jump between different subroutine levels.