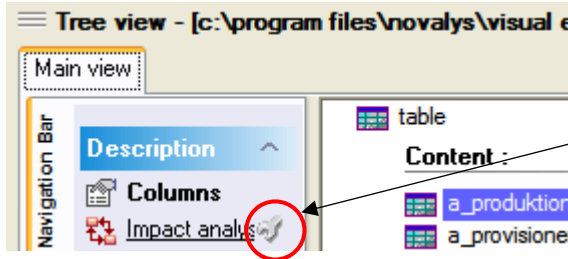


Introduction to Visual Expert 6.0 new features

1. NEW NAVIGATION BAR	2
2. NEW MACROS	3
2.1. LESS MACROS, MORE PARAMETERS	3
2.2. NEW MACROS TO IDENTIFY SPECIFIC DATAWINDOWS	4
2.3. MAPPING DATAWINDOWS / DBMS	4
2.3. MAPPING DATAWINDOWS / DBMS	5
2.4. INHERITANCE HIERARCHY (WINDOWS/UO/MENUS).....	6
3. NEW IMPACT ANALYSIS	7
3.1. PRESENTATION OF THE RESULT	7
3.1.1. <i>In the treeview</i>	7
3.1.2. <i>In the source code view</i>	8
3.2. IMPACT ANALYSIS EXAMPLES	9
3.2.1. <i>Impact Analysis on a Table</i>	9
3.2.2. <i>Impact Analysis on a Database Column</i>	13
3.2.3. <i>Impact Analysis on a Stored Procedure</i>	16
4. EXPLORING REFERENCES WITH VISUAL EXPERT 6.0	18
4.1. REFERENCES FROM PB TO THE DBMS.....	18
4.1.1. <i>Datawindow DataSource</i>	18
4.1.2. <i>Tables and Columns referenced by a DataWindow DataSource</i>	20
4.1.3. <i>Procedures referenced by a DataWindow DataSource</i>	21
4.1.4. <i>Tables and Columns referenced by a PowerScript</i>	22
4.1.5. <i>Other PowerBuilder References</i>	26
4.1.5.1. <i>Cursors and Procedures declared as members of a Component</i>	26
4.1.5.2. <i>Cursors and Procedures declared in a PowerScript</i>	29
4.1.5.3. <i>RPC FUNC</i>	31
4.2. REFERENCES BETWEEN PL/SQL AND T-SQL COMPONENTS	33
4.2.1. <i>Oracle Package</i>	34
4.2.2. <i>Stored Procedures</i>	40
4.2.3. <i>Other DB Code items</i>	41
4.2.4. <i>PL/SQL %TYPE references</i>	41
4.2.5. <i>Parameters & Local Variables</i>	41
5. TOOLTIPS IN THE CODE	42
5.1. POWERBUILDER OBJECTS	43
5.2. POWERBUILDER CONTROLS.....	44
5.3. DATAWINDOWS.....	45
5.4. POWERBUILDER VARIABLES	46
5.5. POWERBUILDER METHODS.....	47
5.6. ORACLE, T-SQL AND DATABASE OBJECTS.....	48
6. NEW SEARCH FEATURE IN THE TREEVIEW	49
6.1. QUICK SEARCH IN THE TREEVIEW	49
7. MISCELLANEOUS	50

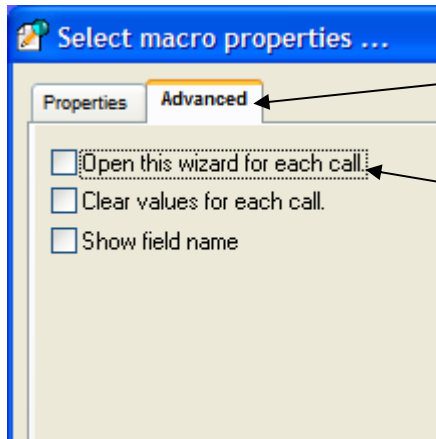
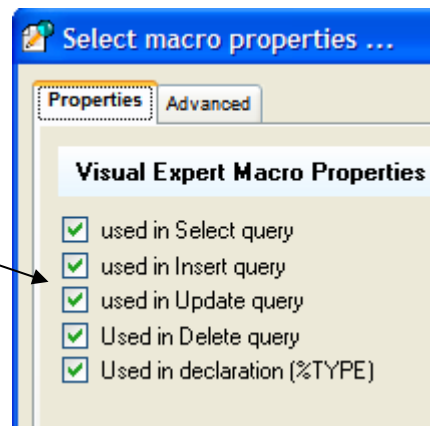
1. New navigation bar

With Visual Expert 6.0, some treeview macros include parameters. These parameters let you adjust the result expected from the Macro.



When parameters are available, a new icon appears when the mouse selects the macro.

If you click on this icon, a dialog box opens to let you check options to adjust the Macro.



A second Tab defines how the Macro will be executed

If you check this box, this dialog box will always open before the Macro is executed

2. New Macros

2.1. Less Macros, more parameters

All treeview macros have been redeveloped and reorganized

A few standard macros are now available for all languages (PB, PL/SQL, and T-SQL):

- Same concept and same Macro name from one language to another
- Each macro uses parameters to cover most needs
- These parameters depend on the type of component selected in the treeview

The screenshot displays the treeview interface for Transact-SQL. The left pane shows the treeview with macros: Definition, References, Impact analysis, Called hierarchy, and Calling hierarchy. The right pane shows the component lists for PowerBuilder, Oracle PL/SQL, and Transact SQL. Callouts provide detailed explanations for the macros:

- Definition:** details about the selected component (methods, variables, ancestor, SQL query, parameters...)
- References:** lists the items *called by* (referenced by) the selected component
- Impact Analysis:** lists the items *calling* (referencing) the selected component
- Called Hierarchy:** chain of methods calling each other in the application

Component list for PowerBuilder :	Count
Datawindow	54 components
PBL	13 components
Application object	1 component
Function object	8 components
Menu	17 components
Window	73 components
userobject	410 components
Structure	41 components

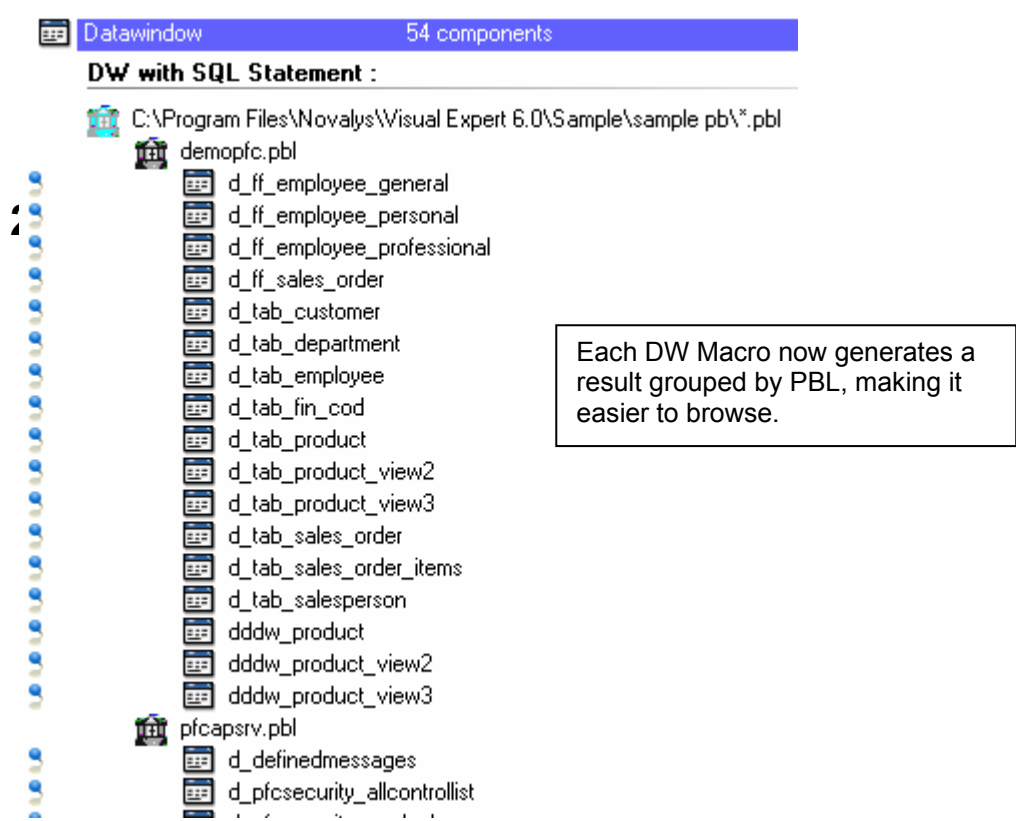
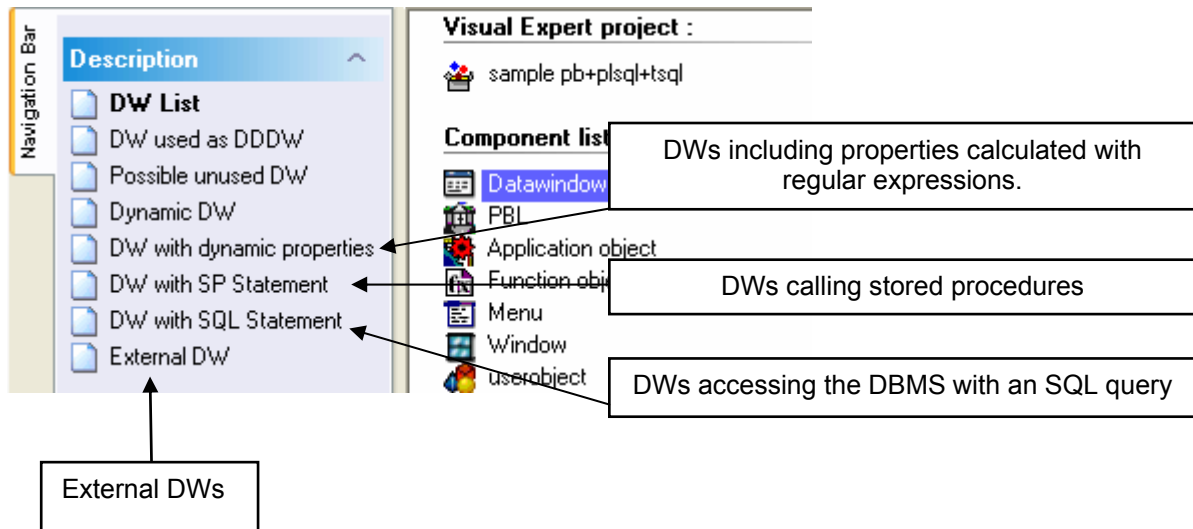
Oracle PL/SQL :	Count
PL/SQL File	6 components
Package	2 components
Stored procedure	15 components
Trigger	1 component
Cursor	2 components

Transact SQL :	Count
T-SQL File (ASE)	6 components
Stored procedure	9 components

Stored proc. list :
OrdersbyEmployee
Productsbycustomer
sp_customer_list
sp_deleteemployee

2.2. New macros to identify specific DataWindows

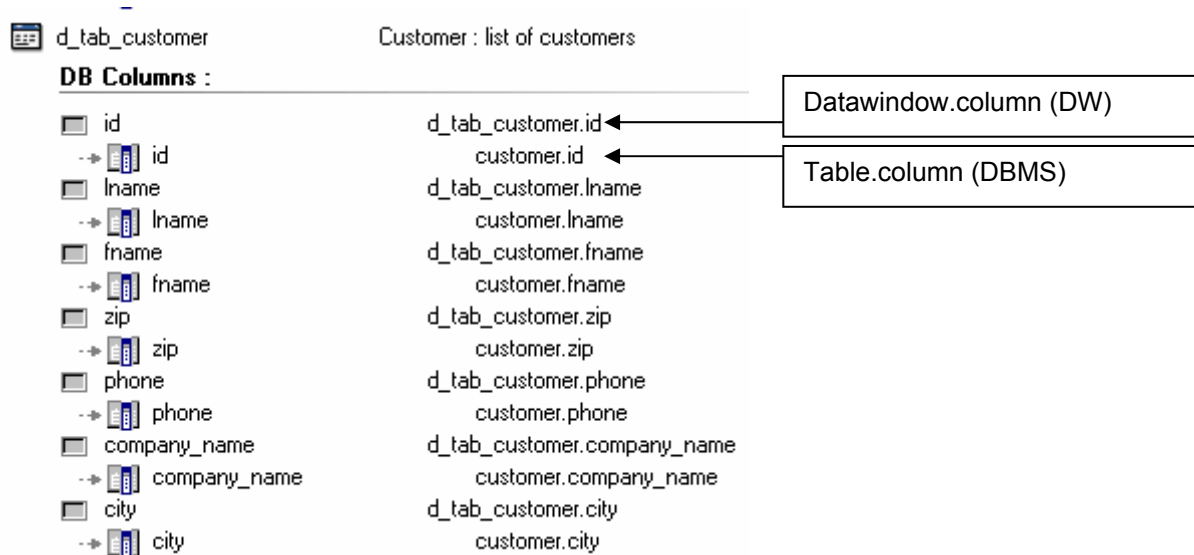
Several new macros have been added with Version 6.0, available at the root of the DWs:



Mapping DataWindows / DBMS

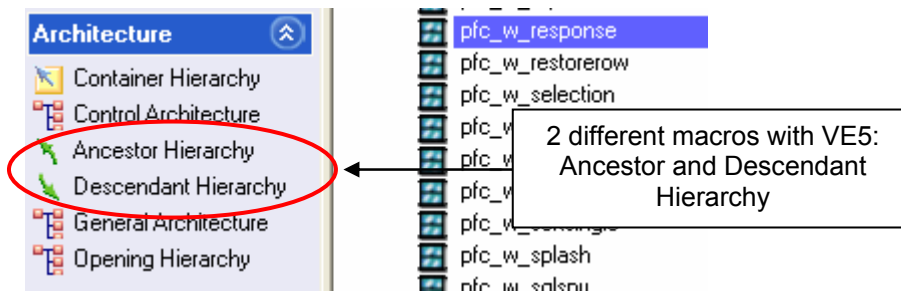
Visual Expert 6.0 also includes a new “DB columns” macro for DataWindows.

This macro displays the DW- DBMS mapping: it shows which table.column in the DBMS corresponds to each DW column.

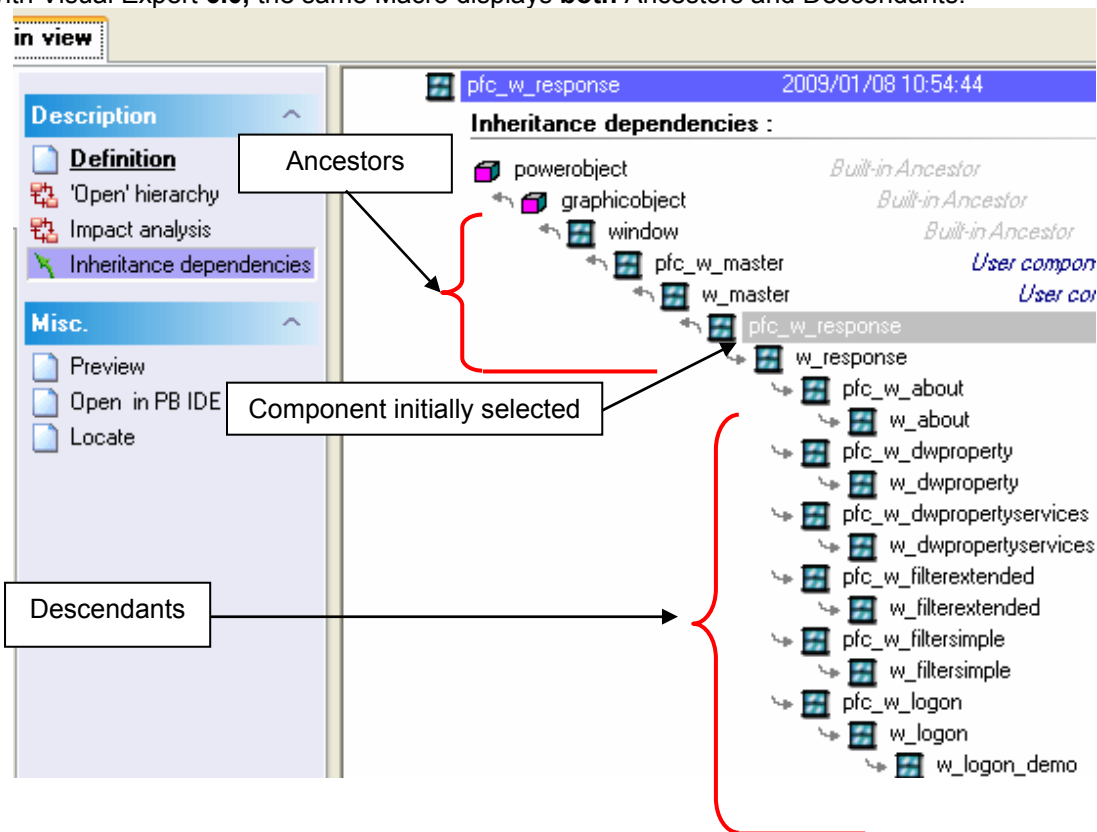


2.4. Inheritance hierarchy (Windows/UO/Menus)

Visual Expert 5.x offered 2 different macros: Descendants + Ancestors



With Visual Expert 6.0, the same Macro displays **both** Ancestors and Descendants:



3. New Impact Analysis

3.1. Presentation of the result

3.1.1. In the treeview

Impact Analysis has been redeveloped for a simpler use.

The macro « impact analysis» now provides a treeview as a result.

Each node in this treeview is either:

- One of the searched items (part of the result of the impact analysis)
- A parent of a searched item

Each searched item comes with a “pin” icon, and a symbol of its dependency with its parent. A parent does not have such icon and symbol.

Example: Impact Analysis on the table « bonus »

SQL Items :

database	2 components
user	7 components
table	43 components

Content :

bonus

Impact analysis :

- validation_pb
- w_sql
 - ue_sql (pin icon, dependency symbol) → This symbol means that this event is referencing the table « Bonus »
 - ue_sql_test (pin icon)
 - bonus_cursor (pin icon, dependency symbol) → This icon means that this item belongs to the Impact Analysis result
 - ue_declare
 - bonus_cursor (dependency symbol)
- d_proc3
- data source
- d_bonus
- data source
- S:\WELOG\2008-11-07 15-57-44 [VITERRA] [WILLIAM SIGUE
- Test_Package
 - proc_SQL
 - [...] {BEGIN SELECT a,b,c,d FROM ttt; proc2(proc

3.1.2. In the source code view

When browsing the result of an Impact Analysis, Visual Expert highlights each reference in the source code view:

As usual, the source code view displays the code of the item selected in the treeview. For instance, if a PL/SQL Package is selected in the treeview, the source code of this package is displayed and all instructions matching the Impact Analysis are highlighted.

Each line number containing such an instruction is listed below the source code view. If you click on one line number, the view will scroll down to this line.

The image shows two screenshots of the Visual Expert interface, illustrating how Impact Analysis results are displayed in the source code view based on the selection in the treeview.

Top Screenshot: The treeview shows a file containing two packages selected. The source code view displays the code for the selected procedure, with all instructions referencing the procedure « Proc 1 » listed below the code. A callout box explains: "This example shows an Impact Analysis for the PLSQL Procedure « Proc 1 ». When selecting a file containing 2 Packages, all instructions referencing the procedure « Proc 1 » are listed below the source code." The source code includes a DECLARE section, a BEGIN section with a call to proc3, and an END section. A list of line numbers (54, 86, 88, 104, 109, 112, 126, 128, 132, 153, 156) is shown below the code, with an arrow pointing to line 112.

Bottom Screenshot: The treeview shows a specific PL/SQL Block selected. The source code view displays the code for this block, with only the 2 references included in this block listed below the code. A callout box explains: "If we select a PLSQL Block in the treeview, only 2 references included in this block are listed." The source code includes a DECLARE section, a BEGIN section with a SELECT statement, and an END section. A list of line numbers (109, 112) is shown below the code, with an arrow pointing to line 112.

3.2. Impact Analysis Examples

3.2.1. Impact Analysis on a Table

Impact Analysis on the table « bonus »

	<pre> 45 string s_member1 46 string sal_var 47 int i_member2 48 49 DECLARE Emp_cur CURSOR FOR 50 SELECT employee.emp_number, employee.emp_r 51 FROM employee 52 WHERE employee.emp_salary > :Sal_var ; 53 54 DECLARE Bonus_Cursor CURSOR FOR 55 Select bonus_date 56 from bonus; 57 58 59 string s_lastMember 60 string s_lastMember2 61 string s_lastMember3 62 </pre>
	<p>Event - event ue_sql_test() from w_sql</p> <pre> 1 string ls1, ls2 2 3 select avg(:ls1), bonus.bonus_id, sysdate, sp_plsql1(), 4 into :ls1, :ls2 5 from myTable2, bonus; 6 7 wf_retrieve_data() 8 9 Select col1, col2, bonus.bonus_date, sp_plsql1(bonus.d 10 into :ls1, :ls2 11 from myTable2, bonus 12 where col2 =col3 13 and f(x) = 100; 14 15 ls2 = ls1 16 17 Select bonus_date,sp_plsql2() 18 into :ls1, :ls2 19 from bonus; 20 21 </pre>

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Impact analysis' tree is expanded to show the 'validation_pb' folder. Under this folder, the 'd_proc3' folder is selected, which contains a 'data source' folder. The right pane shows the 'Dwdatasource - d_proc3.data source inherited from dwdata' window, displaying a SQL query:

```
1 SELECT
2   bonus.bonus_amount,
3   bonus.bonus_date,
4   bonus.emp_id
5 FROM
6   bonus
```

bonus

Impact analysis :

- validation_pb
 - w_sql
 - ue_sql
 - ue_sql_test
 - bonus_cursor
 - ue_declare
 - bonus_cursor
 - d_proc3
 - data source
 - d_bonus
 - data source
- S:\WELOG\2008-11-07 15-57-4
 - Test_Package
 - proc_SQL**
 - {BEGIN SELEC

Plsqlproc - proc_SQL inherited from plsqlproc

← →

```

10
11  PROCEDURE proc_SQL
12
13  AS
14
15      l_varlocal_1 myTable2.col18%type;
16      l_varlocal_2 table1.colXXX%type;
17
18
19  BEGIN
20      SELECT
21          proc1(xx_pref_database.p
22          myTable2.col2,
23          myTable2.col3 + proc2(cc
24          myTable2.col2,
25          myTable2.col3 + proc2(cc
26          myTable2.col2,
27          myTable2.xx_col3 + proc2
28          f(col10,col11 + 10 + xxx
29
30      FROM
31          table1,
32          table2 as x,
33          myTable2
34
35      ;
36      proc1( proc2() );
37      proc4( proc5() );
38
39      sp_plsql2();
40
41  BEGIN
42
43      SELECT a,b,c,d
44      FROM ttt;
45
46      proc2( proc3( proc1() ) );
47
48      Select
49          bonus_date,
50          sp_plsql2()
51      from
52          bonus;
53
54  END ;
55
56      Select bonus_date
57      from bonus;
58
59
60  END;
61

```

bonus

Impact analysis :

- validation_pb
 - w_sql
 - ue_sql
 - ue_sql_test
 - bonus_cursor
 - ue_declare
 - bonus_cursor
 - d_proc3
 - data source
 - d_bonus
 - data source
 - S:\WELOG\2008-11-07 15-57-4
 - Test_Package
 - proc_SQL
 - {BEGIN SELEC

Plsqlblock - {BEGIN SELECT a,b,c,d FROM ttt; proc2(proc3

Find

```

10
11 PROCEDURE proc_SQL
12
13 AS
14
15     l_varlocal_1 myTable2.col18%type;
16     l_varlocal_2 table1.colXXX%type;
17
18
19 BEGIN
20     SELECT
21         proc1(xx_pref_database.p
22         myTable2.col2,
23         myTable2.col3 + proc2(cc
24         myTable2.col2,
25         myTable2.col3 + proc2(cc
26         myTable2.col2,
27         myTable2.xx_col3 + proc2
28         f(col10,col11 + 10 + xxx
29
30 FROM
31     table1,
32     table2 as x,
33     myTable2
34
35 ;
36 proc1( proc2() );
37 proc4( proc5() );
38
39 sp_plsql2();
40
41 BEGIN
42
43     SELECT a,b,c,d
44     FROM ttt;
45
46     proc2( proc3( proc1() ) );
47
48     Select
49         bonus_date,
50         sp_plsql2()
51     from
52         bonus;
53
54 END ;
55
56 Select bonus_date
57 from bonus;
58
59
60 END;
61

```

3.2.2. Impact Analysis on a Database Column

Impact Analysis on the column « bonus.bonus_date »

<p>bonus</p> <p>Columns :</p> <ul style="list-style-type: none"> bonus_amount bonus_date <p>Impact analysis :</p> <ul style="list-style-type: none"> validation_pb <ul style="list-style-type: none"> w_sql <ul style="list-style-type: none"> ue_sql ue_sql_test bonus_cursor ue_declare bonus_cursor d_proc3 <ul style="list-style-type: none"> data source bonus_date d_bonus <ul style="list-style-type: none"> data source bonus_date S:\WELOG\2008-11-07 15-57-44 [v] <ul style="list-style-type: none"> Test_Package <ul style="list-style-type: none"> proc_SQL {BEGIN SELECT a,} 	<p>Event - event ue_sql_test() from w_sql</p> <pre> 1 string ls1, ls2 2 3 select avg(:ls1), bonus.bonus_id, sysdat 4 into :ls1, :ls2 5 from myTable2, bonus; 6 7 wf_retrieve_data() 8 9 ▶ Select col1, col2, bonus.bonus_date, sp_ 10 into :ls1, :ls2 11 from myTable2, bonus 12 where col2 = col3 13 and f(x) = 100; 14 15 ls2 = ls1 16 17 ▶ Select bonus_date, sp_plsql2() 18 into :ls1, :ls2 19 from bonus; 20 </pre>
<p>bonus</p> <p>Columns :</p> <ul style="list-style-type: none"> bonus_amount bonus_date <p>Impact analysis :</p> <ul style="list-style-type: none"> validation_pb <ul style="list-style-type: none"> w_sql <ul style="list-style-type: none"> ue_sql ue_sql_test bonus_cursor ue_declare bonus_cursor d_proc3 <ul style="list-style-type: none"> data source bonus_date d_bonus <ul style="list-style-type: none"> data source bonus_date S:\WELOG\2008-11-07 15-57-44 [v] <ul style="list-style-type: none"> Test_Package <ul style="list-style-type: none"> proc_SQL {BEGIN SELECT a,} 	<p>Logicalcursorname - bonus_cursor inherited</p> <pre> 1 string ls1, ls2 2 long ln 3 4 DECLARE dept_proc_local PROCEDURE 5 DECLARE Emp_cur_local CURSOR FOR 6 SELECT employee.emp_number, e 7 FROM employee 8 WHERE employee.emp_salary > :l 9 10 ls1 = "OK" 11 12 DECLARE Bonus_Cursor CURSOR FOR 13 ▶ Select bonus_date 14 from bonus; 15 16 dw_1.SetItem(1, "bonus_date", "value1" 17 dw_1.SetItem(1, "exam_xref_info_ref_i 18 19 Select employee.emp_number, employe 20 into :ls1, :ls2 21 from employee; 22 23 </pre>

<p>bonus</p> <p>Columns :</p> <ul style="list-style-type: none"> bonus_amount bonus_date <p>Impact analysis :</p> <ul style="list-style-type: none"> validation_pb <ul style="list-style-type: none"> w_sql <ul style="list-style-type: none"> ue_sql ue_sql_test bonus_cursor <ul style="list-style-type: none"> ue_declare <ul style="list-style-type: none"> bonus_cursor d_proc3 <ul style="list-style-type: none"> data source bonus_date d_bonus <ul style="list-style-type: none"> data source bonus_date S:\WELOG\2008-11-07 15-57-44 [v] <ul style="list-style-type: none"> Test_Package <ul style="list-style-type: none"> proc_SQL {BEGIN SELECT a,} 	<p>Dwdatasource - d_proc3.data source inherit</p> <p>Find</p> <pre> 1 SELECT 2 bonus.bonus_amount, 3 bonus.bonus_date, 4 bonus.emp_id 5 FROM 6 bonus </pre>
<p>bonus</p> <p>Columns :</p> <ul style="list-style-type: none"> bonus_amount bonus_date <p>Impact analysis :</p> <ul style="list-style-type: none"> validation_pb <ul style="list-style-type: none"> w_sql <ul style="list-style-type: none"> ue_sql ue_sql_test bonus_cursor <ul style="list-style-type: none"> ue_declare <ul style="list-style-type: none"> bonus_cursor d_proc3 <ul style="list-style-type: none"> data source bonus_date d_bonus <ul style="list-style-type: none"> data source bonus_date S:\WELOG\2008-11-07 15-57-44 [v] <ul style="list-style-type: none"> Test_Package <ul style="list-style-type: none"> proc_SQL {BEGIN SELECT a,} 	<pre> 40 41 BEGIN 42 43 SELECT a,b,c,d 44 FROM ttt; 45 46 proc2(proc3(p 47 48 Select 49 bonus_date, 50 sp_plsql2() 51 from 52 bonus; 53 54 END ; 55 56 Select bonus_date 57 from bonus; 58 </pre>

Impact Analysis on the column « table1.column1 »

<p>table1</p> <p>Columns :</p> <ul style="list-style-type: none"> column1 <ul style="list-style-type: none"> Impact analysis : S:\WELOG\2008-11 <ul style="list-style-type: none"> Package_Refe <ul style="list-style-type: none"> proc3 {BE0 {DE0 proc1 proc2 Package_Refe <ul style="list-style-type: none"> proc4 S:\WELOG\2008-11 <ul style="list-style-type: none"> Test_Package <ul style="list-style-type: none"> csie_invo <ul style="list-style-type: none"> {BE0 { block_ref <ul style="list-style-type: none"> {BE0 {BE0 column2 	<pre> 93 94 95 PROCEDURE proc3(96 param1 IN table1.column1%TYPE 97 param2 IN table1.column2%TYPE 98) 99 AS 100 l_varlocal1 table1.column1%type; 101 102 BEGIN 103 104 SELECT column1, proc1() 105 FROM table1; 106 107 BEGIN 108 109 SELECT column1, column2, proc1() 110 FROM table1; 111 112 proc2(proc3(proc1())); 113 114 END ; 115 116 DECLARE 117 118 l_varlocal1_inBlock table1.column1%typ 119 l_varlocal2_inBlock table1.column2%typ 120 121 BEGIN 122 123 SELECT column1, column2, l_varlocal_1. 124 FROM table1; 125 126 proc2(proc3(proc1())); 127 128 l_varlocal1_inBlock := proc3(l_varloca 129 130 END ; 131 132 proc2(proc1()); 133 134 END ; 135 </pre>
---	--

3.2.3. Impact Analysis on a Stored Procedure

Impact Analysis on the stored procedure « proc1 »:

Impact analysis when a "high-level" PLSQL Block is selected

proc1	Proced	204	
Impact analysis :		205	BEGIN
validation_pb		206	
d_procstock_2		207	SELECT
data source		208	pref_database.pref_user.pr
S:\WELOG\2008-11-07 15-57-44 [v		209	col2,
Package_Reference_2		210	col3 + col4 + col5,
proc4		211	f(col10,col11 + 10 + xxxxxx
S:\WELOG\2008-11-07 15-57-44 [v		212	
Test_Package		213	FROM table1, table2 as x
csie_invoice_summary_B		214	;
{BEGIN SELECT ...		215	proc1 (proc2 ());
{BEGIN BEGIN		216	BEGIN
{BEGIN S		217	BEGIN
{BEGIN S		218	SELECT column1, column2
{BEGIN S		219	FROM table1;
proc_SQL		220	
{BEGIN SELECT a,		221	proc2 (proc3 (proc1 ()));
{BEGIN SELECT a,		222	END ;
{BEGIN SELECT a,		223	BEGIN
{BEGIN SELECT a,		224	SELECT column1, column2
{BEGIN SELECT a,		225	FROM table1;
{BEGIN SELECT a,		226	
{BEGIN SELECT a,		227	proc2 (proc3 (proc1 ()));
{BEGIN SELECT a,		228	END ;
{BEGIN SELECT a,		229	END ;
{BEGIN SELECT a,		230	SELECT column1
{BEGIN SELECT a,		231	FROM table1;
{BEGIN SELECT a,		232	
{BEGIN SELECT a,		233	proc2 (proc3 (proc1 ()));
{BEGIN SELECT a,		234	
{BEGIN SELECT a,		235	END ;
{BEGIN SELECT a,		236	
{BEGIN SELECT a,		237	SELECT column1
{BEGIN SELECT a,		238	FROM table1;
{BEGIN SELECT a,		239	
{BEGIN SELECT a,		240	END ;
{BEGIN SELECT a,		241	
{BEGIN SELECT a,		242	END ;

Impact analysis when a "lower-level" PLSQL Block is selected

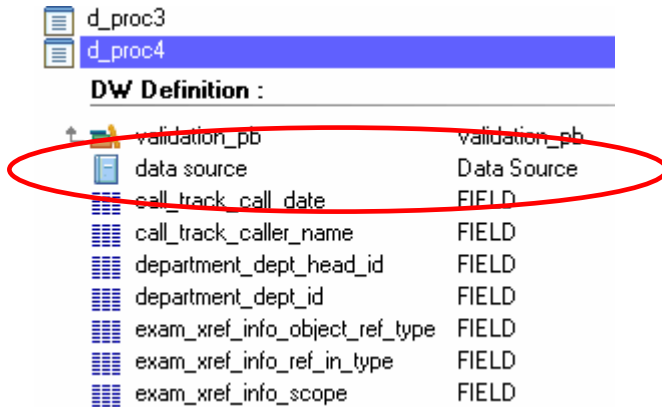
<p>proc1 Proc...</p> <p>Impact analysis :</p> <ul style="list-style-type: none"> validation_pb <ul style="list-style-type: none"> d_procstock_2 <ul style="list-style-type: none"> data source S:\WELOG\2008-11-07 15-57-44 [v] <ul style="list-style-type: none"> Package_Reference_2 <ul style="list-style-type: none"> proc4 S:\WELOG\2008-11-07 15-57-44 [v] <ul style="list-style-type: none"> Test_Package <ul style="list-style-type: none"> csie_invoice_summary_B <ul style="list-style-type: none"> {BEGIN SELECT ...} {BEGIN BEGIN ...} {BEGIN S ...} {BEGIN S ...} proc_SQL <ul style="list-style-type: none"> {BEGIN SELECT a,} TEST1 <ul style="list-style-type: none"> {BEGIN SELECT pr} {BEGIN SELEC block_ref_sample <ul style="list-style-type: none"> {BEGIN SELECT cc {BEGIN SELECT cc 	<p>204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242</p>	<pre> BEGIN SELECT pref_database.pref_user.pr col2, col3 + col4 + col5, f(col10,col11 + 10 + xxxxxx FROM table1, table2 as x ; proc1(proc2()); BEGIN BEGIN SELECT column1, column2 FROM table1; proc2(proc3(proc1())); END ; BEGIN SELECT column1, column2 FROM table1; proc2(proc3(proc1())); END ; SELECT column1 FROM table1; END; </pre>
--	--	--

4. Exploring references with Visual Expert 6.0

4.1. References from PB to the DBMS

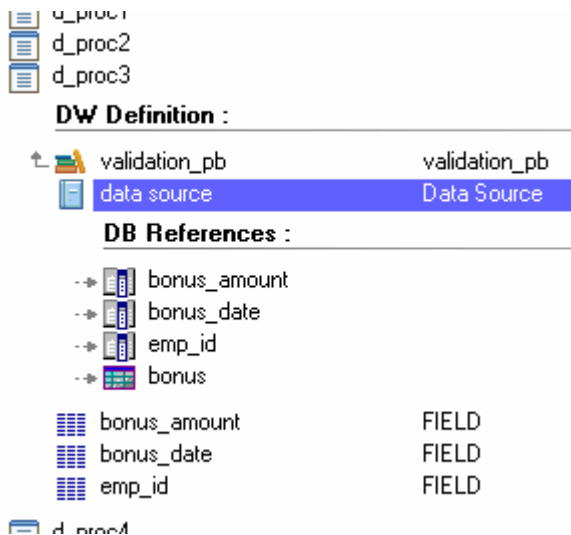
4.1.1. Datawindow DataSource

Visual Expert treeview now displays a Datawindow's « DataSources ».

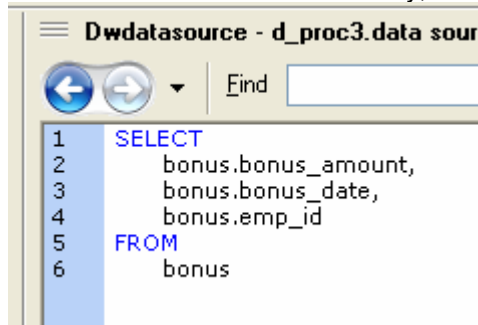


A "DataSource" is displayed when the DW has a data source (SQL Query or Stored Procedure).

The "DataSource" will reference all the items composing the data source (including Update procedures): When the "DataSource" is selected, the macro "reference" displays the type and name of the corresponding Database Objects.



If the DataSource is an SQL Query, this query is displayed in the source code view:



In case of a Stored Procedure, the DataSource references this procedure:

DW Definition :

validation_pb	validation_pb
data source	Data Source

DB References :

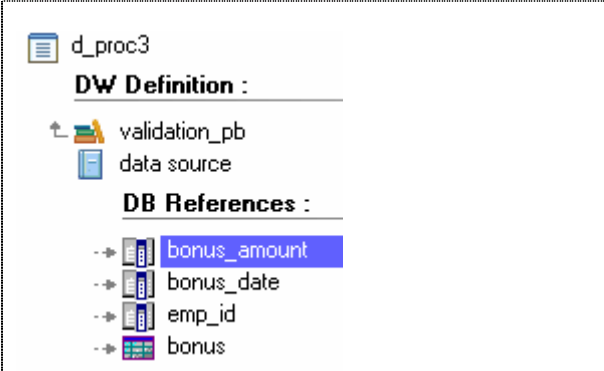
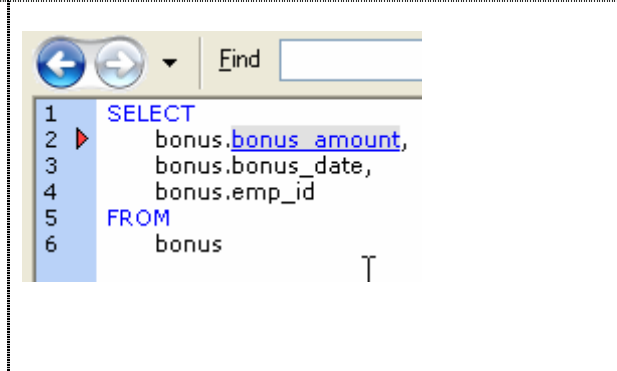
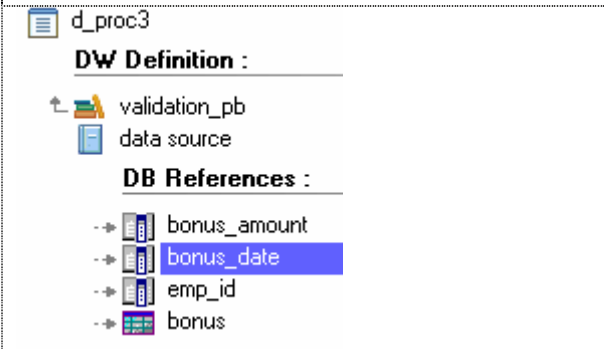
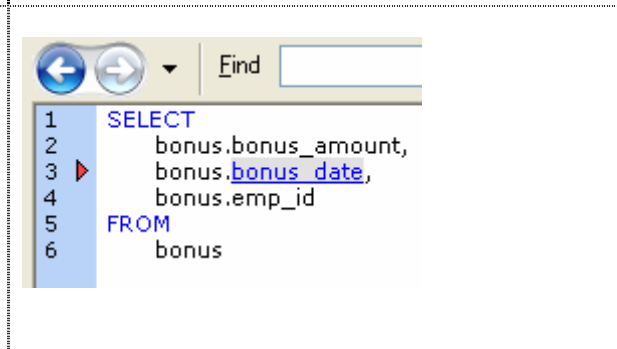
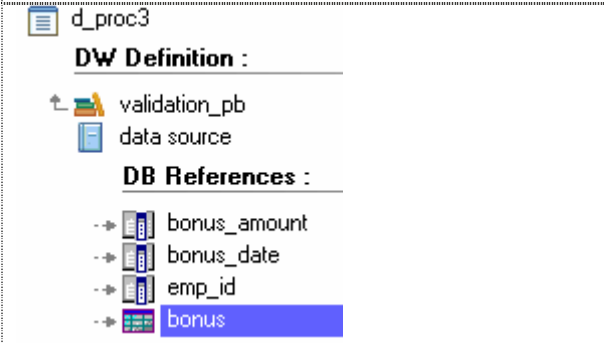
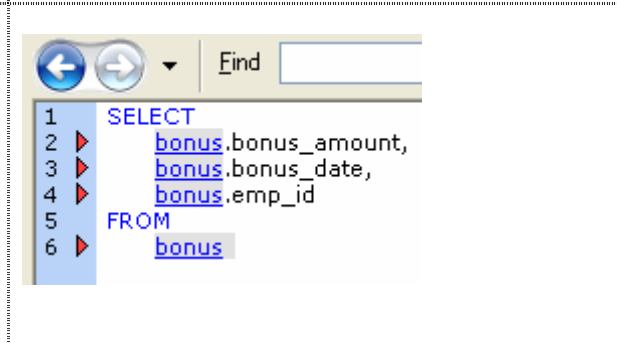
- sp_retrieve_contacts
- city FIELD
- fax FIELD
- first_name FIELD
- id FIELD
- last_name FIELD
- phone FIELD
- state FIELD
- street FIELD
- title FIELD
- zip FIELD

The source code view shows where the procedure is declared in the DW code (export):

```
12 column=(type=char(2) updatewhereclause=yes name=state uu
13 column=(type=char(5) updatewhereclause=yes name=zip dbna
14 column=(type=char(10) updatewhereclause=yes name=phone
15 column=(type=char(10) updatewhereclause=yes name=fax dbr
16 procedure="1 execute dba.sp_retrieve_contacts;0 " )
17 text(band=detail alignment="1" text="Id:" border="0" color="33
18 column(band=detail id=1 alignment="1" tabsequence=32766 boi
19 text(band=detail alignment="1" text="Last Name:" border="0" c
20 column(band=detail id=2 alignment="0" tabsequence=32766 boi
```

4.1.2. Tables and Columns referenced by a DataWindow DataSource.

In case of an SQL DataSource, the table or column selected in the treeview is highlighted in the Source code:

 <p>d_proc3 DW Definition : validation_pb data source DB References : bonus_amount bonus_date emp_id bonus</p>	 <pre>1 SELECT 2 bonus.bonus_amount, 3 bonus.bonus_date, 4 bonus.emp_id 5 FROM 6 bonus</pre>
 <p>d_proc3 DW Definition : validation_pb data source DB References : bonus_amount bonus_date emp_id bonus</p>	 <pre>1 SELECT 2 bonus.bonus_amount, 3 bonus.bonus_date, 4 bonus.emp_id 5 FROM 6 bonus</pre>
 <p>d_proc3 DW Definition : validation_pb data source DB References : bonus_amount bonus_date emp_id bonus</p>	 <pre>1 SELECT 2 bonus.bonus_amount, 3 bonus.bonus_date, 4 bonus.emp_id 5 FROM 6 bonus</pre>

4.1.3. Procedures referenced by a DataWindow DataSource

Both « update » and « select » procedures are referenced by the DataSource:

The screenshot shows the 'd_procstock_multiples' DataWindow. Under 'DW Definition', the 'data source' is listed as 'Data Source'. Under 'DB References', several stored procedures are listed: 'java_debug_detach_from_vm', 'sp_product_info', 'sp_retrieve_contacts', and 'webcopytemplate'.

When a procedure is selected, its declaration is displayed in the DW Source code:

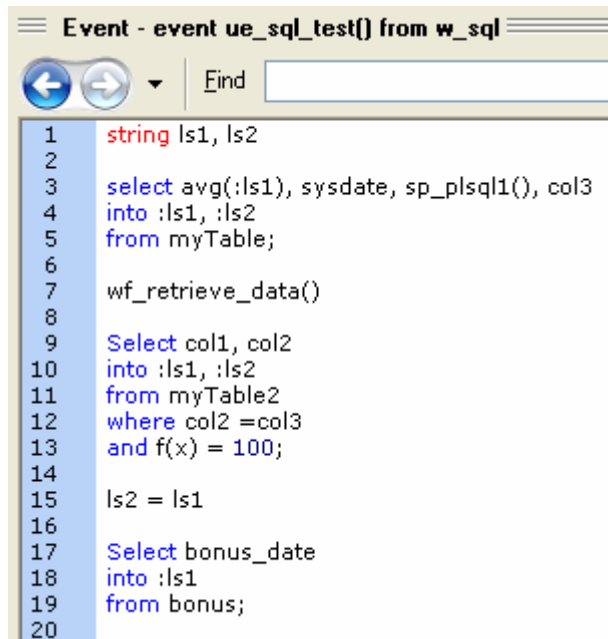
The screenshot shows the 'd_procstock_multiples' DataWindow with 'java_debug_detach_from_vm' selected in the 'DB References' list. The 'DW Source code' pane on the right displays the following SQL code:

```
3 header(height=0 color="536870912" )
4 summary(height=0 color="536870912" )
5 footer(height=0 color="536870912" )
6 detail(height=1368 color="536870912" )
7 table(update.method.type=SP update.method="db
  ↳ =(("Id",column=("id",new,in)),("parentId",unuse
  ↳ "dbo.java_debug_detach_from_vm" insert.metho
  ↳ method.type=SP delete.method="dba.sp_produc
  ↳ unused))column=(type=long updatewhereclause
  ↳ column=(type=char(15) updatewhereclause=yes
```

4.1.4. Tables and Columns referenced by a PowerScript

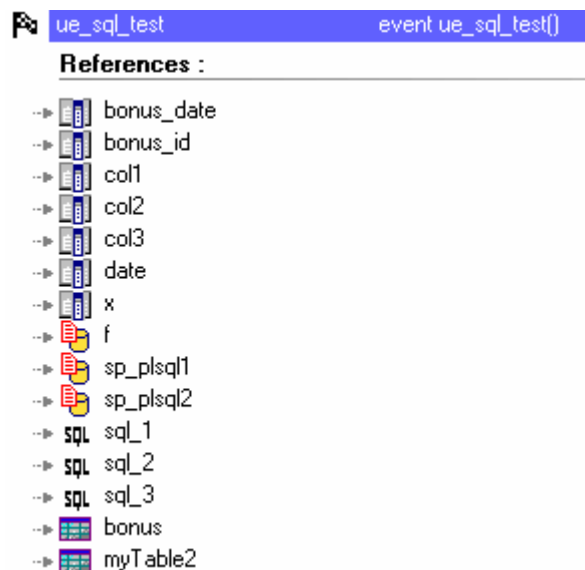
It is now possible to list all DB Objects referenced by a PowerScript

For example, this event includes 3 SQL Queries:



```
1 string ls1, ls2
2
3 select avg(:ls1), sysdate, sp_plsql1(), col3
4 into :ls1, :ls2
5 from myTable;
6
7 wf_retrieve_data()
8
9 Select col1, col2
10 into :ls1, :ls2
11 from myTable2
12 where col2 = col3
13 and f(x) = 100;
14
15 ls2 = ls1
16
17 Select bonus_date
18 into :ls1
19 from bonus;
20
```

You can display the DB Objects referenced by this event...



... and when you select a DB Object , its references are highlighted in the source code:

ue_sql_test

References :

- > bonus_date
- > bonus_id
- > col1
- > col2
- > col3
- > date
- > x
- > f
- > sp_plsql1
- > sp_plsql2
- > sql sql_1
- > sql sql_2
- > sql sql_3
- > **bonus**
- > myTable2

Event - event ue_sql_test() from w_sql

Find Find Next

```

1 string ls1, ls2
2
3 ▶ select avg(:ls1), bonus.bonus_id, sysdate, s
4 into :ls1, :ls2
5 ▶ from myTable2, bonus;
6
7 wf_retrieve_data()
8
9 ▶ Select col1, col2, bonus.bonus_date, sp_pls
10 into :ls1, :ls2
11 ▶ from myTable2, bonus
12 where col2 =col3
13 and f(x) = 100;
14
15 ls2 = ls1
16
17 Select bonus_date,sp_plsql2()
18 into :ls1, :ls2
19 ▶ from bonus;
20
21

```

ue_sql_test

References :

- > **bonus_date**
- > bonus_id
- > col1
- > col2
- > col3
- > date
- > x
- > f
- > sp_plsql1
- > sp_plsql2
- > sql sql_1
- > sql sql_2
- > sql sql_3
- > bonus
- > myTable2

Event - event ue_sql_test() from w_sql

Find Find Next

```

1 string ls1, ls2
2
3 select avg(:ls1), bonus.bonus_id, sysdate, s
4 into :ls1, :ls2
5 from myTable2, bonus;
6
7 wf_retrieve_data()
8
9 ▶ Select col1, col2, bonus.bonus_date, sp_pls
10 into :ls1, :ls2
11 from myTable2, bonus
12 where col2 =col3
13 and f(x) = 100;
14
15 ls2 = ls1
16
17 ▶ Select bonus_date,sp_plsql2()
18 into :ls1, :ls2
19 from bonus;
20
21

```

ue_sql_test

References :

- > bonus_date
- > bonus_id
- > col1
- > col2
- > col3
- > date
- > x
- > f
- > sp_plsql1
- > sp_plsql2
- > SQL sql_1
- > SQL sql_2
- > SQL sql_3
- > bonus
- > myTable2

Event - event ue_sql_test() from w_sql

Find Find Next

```

1 string ls1, ls2
2
3 select avg(:ls1), bonus.bonus_id, sysdate, s
4 into :ls1, :ls2
5 from myTable2, bonus;
6
7 wf_retrieve_data()
8
9 Select col1, col2, bonus.bonus_date, sp_pls
10 into :ls1, :ls2
11 from myTable2, bonus
12 where col2 =col3
13 and f(x) = 100;
14
15 ls2 = ls1
16
17 Select bonus_date,sp_plsql2()
18 into :ls1, :ls2
19 from bonus;
20
21

```

ue_sql_test

References :

- > bonus_date
- > bonus_id
- > col1
- > col2
- > col3
- > date
- > x
- > f
- > sp_plsql1
- > sp_plsql2
- > SQL sql_1
- > SQL sql_2
- > SQL sql_3
- > bonus
- > myTable2

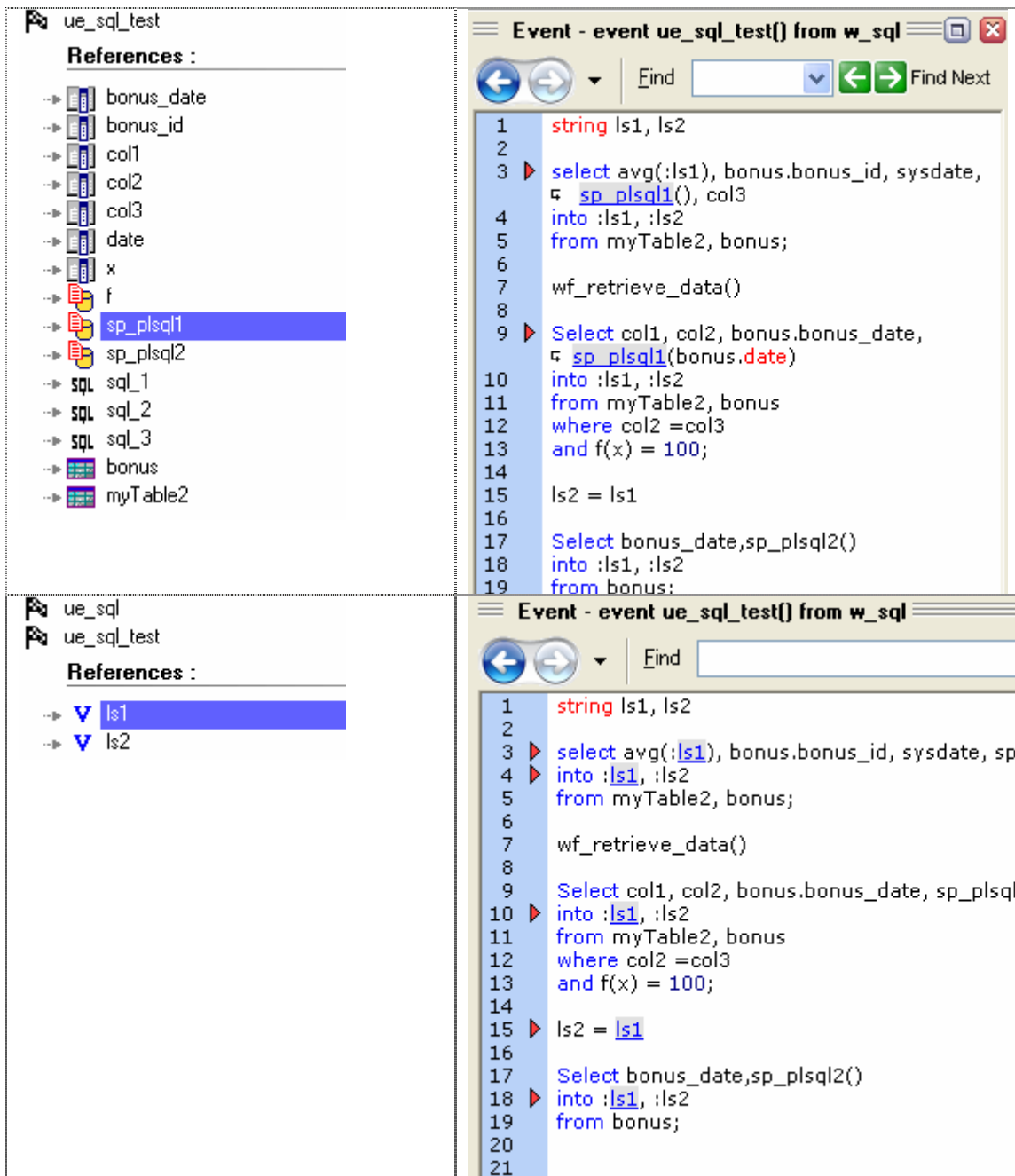
Event - event ue_sql_test() from w_sql

Find Find Next

```

1 string ls1, ls2
2
3 select avg(:ls1), bonus.bonus_id, sysdate, s
4 into :ls1, :ls2
5 from myTable2, bonus;
6
7 wf_retrieve_data()
8
9 Select col1, col2, bonus.bonus_date, sp_pls
10 into :ls1, :ls2
11 from myTable2, bonus
12 where col2 =col3
13 and f(x) = 100;
14
15 ls2 = ls1
16
17 Select bonus_date,sp_plsql2()
18 into :ls1, :ls2
19 from bonus;
20
21

```



Please Note:

- Visual Expert localizes an SQL Query at the first keyword of the Query.
- Visual Expert also highlights references to variables (local, instance, global) in PowerScripts, as well as in SQL Queries.

4.1.5. Other PowerBuilder References

PowerBuilder can also reference DB Objects with « Logical cursor », « Logical procedure » and « RPCFunc » declarations.

3 possibilities:

1. Declaration of a member of a PowerBuilder component (**DECLARE CURSOR** or **DECLARE PROCEDURE**)
2. Declaration in PowerScript (**DECLARE CURSOR** or **DECLARE PROCEDURE**)
3. Declaration of a PowerBuilder method with the instruction **RPCFUNC**

4.1.5.1. Cursors and Procedures declared as members of a Component

In this case, Visual Expert creates « LogicalCursor » and « LogicalProcedure » items. « LogicalCursor » and « LogicalProcedure » are contained in the component (like its controls). As for a DW DataSource they are directly referencing DB Objects.

For example, this window includes 2 logicalCursors and 3 LogicalProcedures:

Definition :	
validation_pb	N:\Projets_FA\validation_pb.pbl
cb_1	w_sql.cb_1
dw_1	w_sql.dw_1
bonus_cursor	w_sql.bonus_cursor
emp_cur	w_sql.emp_cur
dept_proc	w_sql.dept_proc
dept_proc_2	w_sql.dept_proc_2
dept_proc_3	w_sql.dept_proc_3

When a « LogicalCursor » or a « LogicalProcedure » is selected, its definition is highlighted in the source code view:

<p>w_sql</p> <p>Definition :</p> <ul style="list-style-type: none"> validation_pb cb_1 dw_1 bonus_cursor emp_cur dept_proc dept_proc_2 dept_proc_3 dept i_member2 s_lastmember s_lastmember2 s_lastmember3 	<pre> 26 dw_1 dw_1 27 cb_1 cb_1 28 end type 29 global w_sql w_sql 30 31 type variables 32 33 34 // 35 string dept 36 ▶ DECLARE dept_proc PROCEDURE FOR pref_user.spm1(:dept); 37 38 DECLARE dept_proc_2 PROCEDURE FOR spm1(:dept); 39 40 DECLARE dept_proc_3 PROCEDURE FOR 41 pref_user . spm2 (:dept); 42 43 // 44 </pre>
<p>w_sql</p> <p>Definition :</p> <ul style="list-style-type: none"> validation_pb cb_1 dw_1 bonus_cursor emp_cur dept_proc dept_proc_2 dept_proc_3 dept i_member2 s_lastmember s_lastmember2 s_lastmember3 	<pre> 39 40 DECLARE dept_proc_3 PROCEDURE FOR 41 pref_user . spm2 (:dept); 42 43 // 44 45 string s_member1 46 string sal_var 47 int i_member2 48 49 ▶ DECLARE Emp_cur CURSOR FOR 50 SELECT employee.emp_number, employee.emp_name 51 FROM employee 52 WHERE employee.emp_salary > :Sal_var ; 53 54 DECLARE Bonus_Cursor CURSOR FOR 55 Select bonus_date 56 from bonus; 57 </pre>
<p>w_sql</p> <p>Definition :</p> <ul style="list-style-type: none"> validation_pb cb_1 dw_1 bonus_cursor emp_cur dept_proc dept_proc_2 dept_proc_3 dept i_member2 s_lastmember s_lastmember2 s_lastmember3 	<pre> 44 45 string s_member1 46 string sal_var 47 int i_member2 48 49 DECLARE Emp_cur CURSOR FOR 50 SELECT employee.emp_number, employee.emp_name 51 FROM employee 52 WHERE employee.emp_salary > :Sal_var ; 53 54 ▶ DECLARE Bonus_Cursor CURSOR FOR 55 Select bonus_date 56 from bonus; 57 58 59 string s_lastMember 60 string s_lastMember2 61 string s_lastMember3 62 </pre>

Visual Expert displays the DB Objects referenced by LogicalCursors and LogicalProcedures.
The corresponding reference is highlighted in the source code:

<p>w_sql</p> <p>Definition :</p> <ul style="list-style-type: none"> validation_pb <ul style="list-style-type: none"> cb_1 dw_1 bonus_cursor emp_cur dept_proc Stored procedure r <ul style="list-style-type: none"> spm1 dept_proc_2 dept_proc_3 dept i_member2 	<pre> 31 type variables 32 33 34 // 35 string dept 36 ▶ DECLARE dept_proc PROCEDURE FOR pref_user.spm1(:dept) 37 38 DECLARE dept_proc_2 PROCEDURE FOR spm1(:dept); 39 40 DECLARE dept_proc_3 PROCEDURE FOR 41 pref_user . spm2 (:dept); 42 43 // 44 45 string s_member1 46 string sal_var 47 int i_member2 48 </pre>
<ul style="list-style-type: none"> bonus_cursor emp_cur DB items reference <ul style="list-style-type: none"> emp_name emp_number emp_salary employee dept_proc dept_proc_2 dept_proc_3 dept i_member2 s_lastmember s_lastmember2 s_lastmember3 	<pre> 45 string s_member1 46 string sal_var 47 int i_member2 48 49 DECLARE Emp_cur CURSOR FOR 50 ▶ SELECT employee.emp_number, employee.emp_name 51 ▶ FROM employee 52 ▶ WHERE employee.emp_salary > :Sal_var ; 53 54 DECLARE Bonus_Cursor CURSOR FOR 55 Select bonus_date 56 from bonus; 57 58 59 string s_lastMember 60 string s_lastMember2 61 string s_lastMember3 62 </pre>

4.1.5.2. Cursors and Procedures declared in a PowerScript

The macro “references” displays the logical cursors and procedures declared in PowerScript. For example:

ue_declare event ue_declare()

References :

- > **a** dept string
- > **v** ls1 string
- > **f(x)** setitem DATAWINDOWCONTROL
- > **dw_1**
- > **bonus_date**
- > **exam_xref_info_ref_in_type**
- > **SQL** **bonus_cursor**
- > **SQL** **emp_cur_local**
- > **SP** **dept_proc_local**

Again, the declaration is highlighted in the source code:

<p>ue_declare</p> <p>References :</p> <ul style="list-style-type: none"> -> a dept -> v ls1 -> f(x) setitem -> dw_1 -> bonus_date -> exam_xref_info_ref_in_type -> SQL bonus_cursor -> SQL emp_cur_local -> SP dept_proc_local 	<p>Event - event ue_declare() from w_sql</p> <pre> 1 string ls1 2 long ln 3 4 DECLARE dept_proc_local PROCEDURE FOR spm_local(:dept); 5 DECLARE Emp_cur_local CURSOR FOR 6 SELECT employee.emp_number, employee.emp_name 7 FROM employee 8 WHERE employee.emp_salary > :Sal_var ; 9 10 ls1 = "Ok" 11 12 ▶ DECLARE Bonus_Cursor CURSOR FOR 13 Select bonus_date 14 from bonus; 15 16 dw_1.SetItem(1, "bonus_date", "coucou") 17 dw_1.SetItem(1, "exam_xref_info_ref_in_type", "coucou") 18 </pre>
<p>ue_declare</p> <p>References :</p> <ul style="list-style-type: none"> -> a dept -> v ls1 -> f(x) setitem -> dw_1 -> bonus_date -> exam_xref_info_ref_in_type -> SQL bonus_cursor -> SQL emp_cur_local -> SP dept_proc_local 	<p>Event - event ue_declare() from w_sql</p> <pre> 1 string ls1 2 long ln 3 4 DECLARE dept_proc_local PROCEDURE FOR spm_local(:dept); 5 ▶ DECLARE Emp_cur_local CURSOR FOR 6 SELECT employee.emp_number, employee.emp_name 7 FROM employee 8 WHERE employee.emp_salary > :Sal_var ; 9 10 ls1 = "Ok" 11 12 DECLARE Bonus_Cursor CURSOR FOR 13 Select bonus_date 14 from bonus; 15 16 dw_1.SetItem(1, "bonus_date", "coucou") 17 dw_1.SetItem(1, "exam_xref_info_ref_in_type", "coucou") 18 </pre>

ue_declare

References :

- > a dept
- > v ls1
- > f(x) setitem
- > dw_1
- > bonus_date
- > exam_xref_info_ref_in_type
- > SQL bonus_cursor
- > SQL emp_cur_local
- > SP dept_proc_local

Event - event ue_declare() from w_sql

```

1 string ls1
2 long ln
3
4 DECLARE dept_proc_local PROCEDURE FOR spm_local(:dept);
5 DECLARE Emp_cur_local CURSOR FOR
6     SELECT employee.emp_number, employee.emp_name
7     FROM employee
8     WHERE employee.emp_salary > :Sal_var ;
9
10 ls1 = "ok"
11
12 DECLARE Bonus_Cursor CURSOR FOR
13     Select bonus_date
14     from bonus;
15
16 dw_1.SetItem(1, "bonus_date", "coucou")
17 dw_1.SetItem(1, "exam_xref_info_ref_in_type", "coucou")
18

```

Visual Expert can display the DB Objects references by logical cursors and procedures:
The corresponding references are highlighted in the source code:

ue_declare

References :

- > a dept
- > v ls1
- > f(x) setitem
- > dw_1
- > bonus_date
- > exam_xref_info_ref_in_type
- > SQL bonus_cursor
- > SQL emp_cur_local

DB items referenced :

- > emp_name
- > emp_number
- > emp_salary
- > employee

-> SP dept_proc_local

Logicalcursorname - emp_cur_local inherited from logicalcursorname

```

1 string ls1
2 long ln
3
4 DECLARE dept_proc_local PROCEDURE FOR spm_local(:dept);
5 DECLARE Emp_cur_local CURSOR FOR
6     SELECT employee.emp_number, employee.emp_name
7     FROM employee
8     WHERE employee.emp_salary > :Sal_var ;
9
10 ls1 = "ok"
11
12 DECLARE Bonus_Cursor CURSOR FOR
13     Select bonus_date
14     from bonus;
15
16 dw_1.SetItem(1, "bonus_date", "coucou")
17 dw_1.SetItem(1, "exam_xref_info_ref_in_type", "coucou")
18

```

ue_declare

References :

- > a dept
- > v ls1
- > f(x) setitem
- > dw_1
- > bonus_date
- > exam_xref_info_ref_in_type
- > SQL bonus_cursor
- > SQL emp_cur_local
- > SP dept_proc_local

Stored procedure refere

- > spm_local

Logicalprocname - dept_proc_local inherited from logicalprocname

```

1 string ls1
2 long ln
3
4 DECLARE dept_proc_local PROCEDURE FOR spm_local(:dept);
5 DECLARE Emp_cur_local CURSOR FOR
6     SELECT employee.emp_number, employee.emp_name
7     FROM employee
8     WHERE employee.emp_salary > :Sal_var ;
9
10 ls1 = "ok"
11
12 DECLARE Bonus_Cursor CURSOR FOR
13     Select bonus_date
14     from bonus;
15
16 dw_1.SetItem(1, "bonus_date", "coucou")
17 dw_1.SetItem(1, "exam_xref_info_ref_in_type", "coucou")
18

```

4.1.5.3. RPC FUNC

An RPCFUNC declaration is another solution for PowerBuilder to use a stored procedure.

This declaration is similar to the declaration of an external dll function. The RPCFUNC keyword indicates that a stored procedure is called (and not a dll function).

Visual Expert considers RPCFUNC methods like any other method of the component:

w_rpcfunc_test	
Definition :	
validation_pb	N:\Projets_FA\validation_pb.pbl
activate	event activate
clicked	event clicked
close	event close
open	event open
f(s) my_dll_func1	function string my_dll_func1(string, string) library "myDLL.dll" alias for external_func1
f(s) my_dll_func2	function string my_dll_func2(string, string) library "myDLL.dll" alias for external_func2
f(s) my_plsql_proc1	function string my_plsql_proc1(string) rpcfunc alias for sp_plsql1
f(s) my_plsql_proc2	function string my_plsql_proc2(string) rpcfunc alias for sp_plsql2
f(s) uf_method1	public function integer uf_method1(integer)
f(s) uf_method2	public subroutine uf_method2()

When a PowerBuilder event or function is selected, its source code is displayed.

When an RPCFUNC method or dll is selected, Visual Expert displays its declaration:

<table border="1"> <thead> <tr> <th colspan="2">w_rpcfunc_test</th> </tr> <tr> <th colspan="2">Definition :</th> </tr> </thead> <tbody> <tr> <td>validation_pb</td> <td>N:\Projets_FA\validation_pb.pbl</td> </tr> <tr> <td>activate</td> <td>event activate</td> </tr> <tr> <td>clicked</td> <td>event clicked</td> </tr> <tr> <td>close</td> <td>event close</td> </tr> <tr> <td>open</td> <td>event open</td> </tr> <tr> <td>f(s) my_dll_func1</td> <td>function string my_dll_func1(string, string) library "myDLL.dll" alias for external_func1</td> </tr> <tr> <td>f(s) my_dll_func2</td> <td>function string my_dll_func2(string, string) library "myDLL.dll" alias for external_func2</td> </tr> <tr> <td>f(s) my_plsql_proc1</td> <td>function string my_plsql_proc1(string) rpcfunc alias for sp_plsql1</td> </tr> <tr> <td>f(s) my_plsql_proc2</td> <td>function string my_plsql_proc2(string) rpcfunc alias for sp_plsql2</td> </tr> <tr> <td>f(s) uf_method1</td> <td>public function integer uf_method1(integer)</td> </tr> <tr> <td>f(s) uf_method2</td> <td>public subroutine uf_method2()</td> </tr> </tbody> </table>	w_rpcfunc_test		Definition :		validation_pb	N:\Projets_FA\validation_pb.pbl	activate	event activate	clicked	event clicked	close	event close	open	event open	f(s) my_dll_func1	function string my_dll_func1(string, string) library "myDLL.dll" alias for external_func1	f(s) my_dll_func2	function string my_dll_func2(string, string) library "myDLL.dll" alias for external_func2	f(s) my_plsql_proc1	function string my_plsql_proc1(string) rpcfunc alias for sp_plsql1	f(s) my_plsql_proc2	function string my_plsql_proc2(string) rpcfunc alias for sp_plsql2	f(s) uf_method1	public function integer uf_method1(integer)	f(s) uf_method2	public subroutine uf_method2()	<pre> 20 21 type prototypes 22 23 // déclarations de procédures stockées sous forme de méthode (RPCFUNC) 24 // 25 ► function string my_plsql_proc1 (string toto) rpcfunc alias for sp_plsql1 26 function string my_plsql_proc2 (string toto) rpcfunc alias for sp_plsql2 27 28 // déclarations de fonctions externes de DLL 29 // 30 function string my_dll_func1 (string toto1, string toto2) library "myDLL.d 31 function string my_dll_func2 (string toto1, string toto2) library "myDLL.d 32 </pre>
w_rpcfunc_test																											
Definition :																											
validation_pb	N:\Projets_FA\validation_pb.pbl																										
activate	event activate																										
clicked	event clicked																										
close	event close																										
open	event open																										
f(s) my_dll_func1	function string my_dll_func1(string, string) library "myDLL.dll" alias for external_func1																										
f(s) my_dll_func2	function string my_dll_func2(string, string) library "myDLL.dll" alias for external_func2																										
f(s) my_plsql_proc1	function string my_plsql_proc1(string) rpcfunc alias for sp_plsql1																										
f(s) my_plsql_proc2	function string my_plsql_proc2(string) rpcfunc alias for sp_plsql2																										
f(s) uf_method1	public function integer uf_method1(integer)																										
f(s) uf_method2	public subroutine uf_method2()																										
<table border="1"> <thead> <tr> <th colspan="2">w_rpcfunc_test</th> </tr> <tr> <th colspan="2">Definition :</th> </tr> </thead> <tbody> <tr> <td>validation_pb</td> <td>N:\Projets_FA\validation_pb.pbl</td> </tr> <tr> <td>activate</td> <td>event activate</td> </tr> <tr> <td>clicked</td> <td>event clicked</td> </tr> <tr> <td>close</td> <td>event close</td> </tr> <tr> <td>open</td> <td>event open</td> </tr> <tr> <td>f(s) my_dll_func1</td> <td>function string my_dll_func1(string, string) library "myDLL.dll" alias for external_func1</td> </tr> <tr> <td>f(s) my_dll_func2</td> <td>function string my_dll_func2(string, string) library "myDLL.dll" alias for external_func2</td> </tr> <tr> <td>f(s) my_plsql_proc1</td> <td>function string my_plsql_proc1(string) rpcfunc alias for sp_plsql1</td> </tr> <tr> <td>f(s) my_plsql_proc2</td> <td>function string my_plsql_proc2(string) rpcfunc alias for sp_plsql2</td> </tr> <tr> <td>f(s) uf_method1</td> <td>public function integer uf_method1(integer)</td> </tr> <tr> <td>f(s) uf_method2</td> <td>public subroutine uf_method2()</td> </tr> </tbody> </table>	w_rpcfunc_test		Definition :		validation_pb	N:\Projets_FA\validation_pb.pbl	activate	event activate	clicked	event clicked	close	event close	open	event open	f(s) my_dll_func1	function string my_dll_func1(string, string) library "myDLL.dll" alias for external_func1	f(s) my_dll_func2	function string my_dll_func2(string, string) library "myDLL.dll" alias for external_func2	f(s) my_plsql_proc1	function string my_plsql_proc1(string) rpcfunc alias for sp_plsql1	f(s) my_plsql_proc2	function string my_plsql_proc2(string) rpcfunc alias for sp_plsql2	f(s) uf_method1	public function integer uf_method1(integer)	f(s) uf_method2	public subroutine uf_method2()	<pre> 25 function string my_plsql_proc1 (string toto) rpcfunc alias for sp_plsql1 26 function string my_plsql_proc2 (string toto) rpcfunc alias for sp_plsql2 27 28 // déclarations de fonctions externes de DLL 29 // 30 ► function string my_dll_func1 (string toto1, string toto2) library "myDLL.d 31 function string my_dll_func2 (string toto1, string toto2) library "myDLL.d 32 33 34 35 end prototypes 36 forward prototypes 37 public function integer uf_method1 (integer p1) </pre>
w_rpcfunc_test																											
Definition :																											
validation_pb	N:\Projets_FA\validation_pb.pbl																										
activate	event activate																										
clicked	event clicked																										
close	event close																										
open	event open																										
f(s) my_dll_func1	function string my_dll_func1(string, string) library "myDLL.dll" alias for external_func1																										
f(s) my_dll_func2	function string my_dll_func2(string, string) library "myDLL.dll" alias for external_func2																										
f(s) my_plsql_proc1	function string my_plsql_proc1(string) rpcfunc alias for sp_plsql1																										
f(s) my_plsql_proc2	function string my_plsql_proc2(string) rpcfunc alias for sp_plsql2																										
f(s) uf_method1	public function integer uf_method1(integer)																										
f(s) uf_method2	public subroutine uf_method2()																										

Visual Expert also displays the references to RPCFUNC or dll functions when they are selected:

<p>open</p> <p>f(x) my_dll_func1</p> <p>f(x) my_dll_func2</p> <p>f(x) my_plsql_proc1</p> <p>References :</p> <p>→ sp_plsql1</p> <p>f(x) my_plsql_proc2</p> <p>f(x) uf_method1</p> <p>f(x) uf_method2</p>	<pre> 20 21 type prototypes 22 23 // déclarations de procédures stockées sous forme de méthode 24 (RPCFUNC) 25 // 26 ▶ function string my_plsql_proc1 (string toto) rpcfunc alias for sp_plsql1 27 function string my_plsql_proc2 (string toto) rpcfunc alias for sp_plsql2 28 29 // déclarations de fonctions externes de DLL 30 // 31 function string my_dll_func1 (string toto1, string toto2) library </pre>
<p>open</p> <p>f(x) my_dll_func1</p> <p>References :</p> <p>→ f(x) external_func1</p> <p>f(x) my_dll_func2</p> <p>f(x) my_plsql_proc1</p> <p>f(x) my_plsql_proc2</p> <p>f(x) uf_method1</p> <p>f(x) uf_method2</p>	<pre> 24 // 25 function string my_plsql_proc1 (string toto) rpcfunc alias for sp_plsql1 26 function string my_plsql_proc2 (string toto) rpcfunc alias for sp_plsql2 27 28 // déclarations de fonctions externes de DLL 29 // 30 ▶ function string my_dll_func1 (string toto1, string toto2) library 31 "myDLL.dll" alias for external_func1 32 function string my_dll_func2 (string toto1, string toto2) library 33 "myDLL.dll" alias for external_func2 </pre>

4.2. References between PL/SQL and T-SQL Components

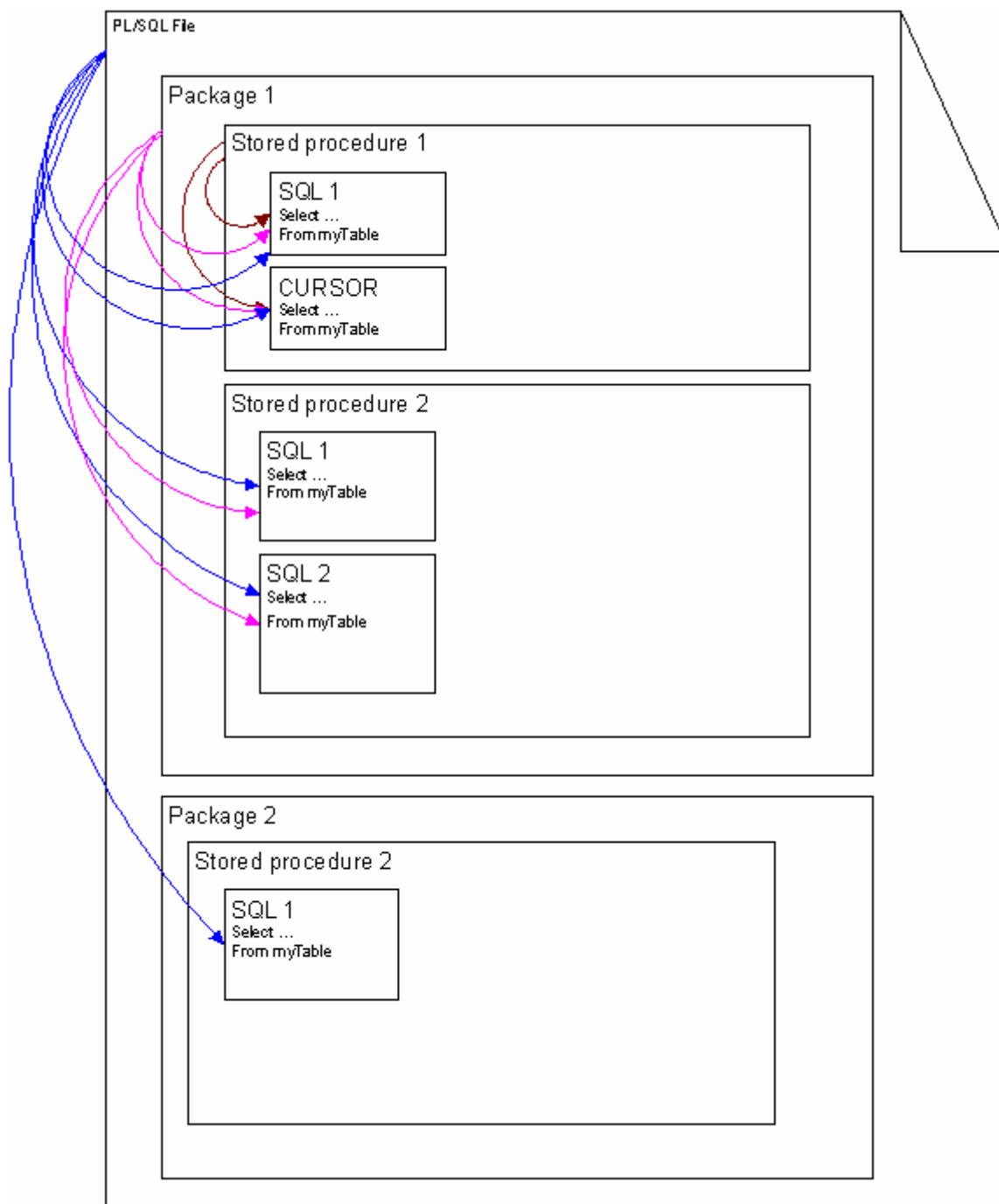
For DB code, Visual Expert does not display SQL items anymore. DB Objects are directly related to PL/SQL or T-SQL objects (packages, procedures, triggers, views, types, blocks...).

PL/SQL blocks (BEGIN ... END) are now related to the DB Objects they reference, as well as the DB Objects referenced in the blocks they include.

Visual Expert can explore a chain of containers to list references at different levels: you can find references in Blocks, Stored Procedures, Packages or Files.

Consequently, you can select a file, a package or a procedure, display the items it references and highlight the corresponding references in the source code.

As shown below, references are accessible at every level of the Code:

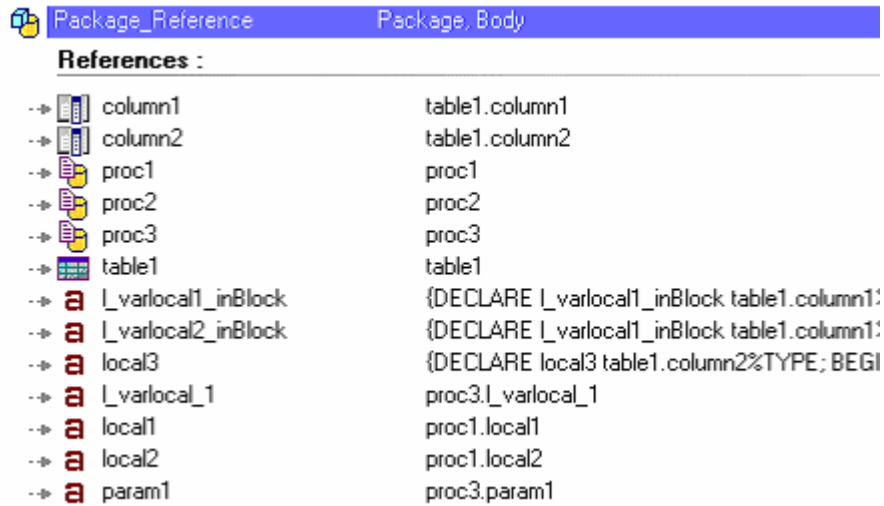


4.2.1. Oracle Package

A Package is now related to all items it references, as well as all items referenced by its stored procedures, types, etc...

As a result, a Package may reference lots of items. You can list all referenced items in the treeview and select one of them: the source code will automatically highlight the references to this item.

For instance:



The screenshot shows a software interface with a blue header bar containing two tabs: 'Package_Reference' and 'Package, Body'. Below the header, the title 'References :' is displayed. A list of references is shown, each with a small icon and a text description. The references include columns, procedures, tables, and local variables, with some showing their full declarations or definitions.

Icon	Reference Name	Reference Path / Declaration
Column	column1	table1.column1
Column	column2	table1.column2
Procedure	proc1	proc1
Procedure	proc2	proc2
Procedure	proc3	proc3
Table	table1	table1
Local Variable	_varlocal1_inBlock	{DECLARE _varlocal1_inBlock table1.column1:
Local Variable	_varlocal2_inBlock	{DECLARE _varlocal1_inBlock table1.column1:
Local Variable	local3	{DECLARE local3 table1.column2%TYPE; BEGI
Local Variable	_varlocal_1	proc3._varlocal_1
Local Variable	local1	proc1.local1
Local Variable	local2	proc1.local2
Local Variable	param1	proc3.param1

<p>Package_Reference</p> <p>References :</p> <ul style="list-style-type: none"> -> column1 -> column2 -> proc1 -> proc2 -> proc3 -> table1 -> l_varlocal1_inBlock -> l_varlocal2_inBlock -> local3 -> l_varlocal_1 -> local1 -> local2 -> param1 	<p>44</p> <p>45</p> <p>46</p> <p>47</p> <p>48 ▶</p> <p>49</p> <p>50</p> <p>51</p> <p>52</p> <p>53</p> <p>54 ▶</p> <p>55</p> <p>56</p> <p>57</p> <p>58</p> <p>59</p> <p>60</p> <p>61</p> <p>62</p> <p>63</p> <p>64</p> <p>65</p> <p>66</p> <p>67</p> <p>68</p> <p>69</p> <p>70</p> <p>71</p> <p>72</p> <p>73</p> <p>74</p> <p>75</p> <p>90</p> <p>91</p> <p>92</p> <p>93</p> <p>94</p> <p>95 ▶</p> <p>96</p> <p>97</p> <p>98</p> <p>99</p> <p>100 ▶</p> <p>101</p> <p>102</p> <p>103</p> <p>104 ▶</p> <p>105</p> <p>106</p> <p>107</p> <p>108</p> <p>109 ▶</p> <p>110</p> <p>111</p> <p>112</p> <p>113</p> <p>114</p> <p>115</p> <p>116</p> <p>117</p> <p>118 ▶</p> <p>119</p> <p>120</p> <p>121</p>	<pre> ----- PROCEDURE proc1 AS local1 table1.column1%TYPE; local2 table1.column2%TYPE; BEGIN SELECT column1, column2, proc1() FROM table1; proc2(); local1 := local2; DECLARE local3 table1.column2%TYPE; BEGIN local1 := local2; local1 := local1 + local2; local3 := local1 + local3; END; END; ----- PROCEDURE proc3(param1 IN table1.column1% param2 IN table1.column2%) AS l_varlocal_1 table1.column1%type; BEGIN SELECT column1, proc1() FROM table1; BEGIN SELECT column1, column2, proc1() FROM table1; proc2(proc3(proc1())); END ; DECLARE l_varlocal1_inBlock table1.column1 l_varlocal2_inBlock table1.column2 BEGIN </pre>
---	--	--

<p>Package_Reference</p> <p>References :</p> <ul style="list-style-type: none"> -> column1 -> column2 -> proc1 -> proc2 -> proc3 -> table1 -> L_varlocal1_inBlock -> L_varlocal2_inBlock -> local3 -> L_varlocal_1 -> local1 -> local2 -> param1 	<pre> 42 IS 43 44 ----- 45 PROCEDURE proc1 46 AS 47 48 local1 table1.column1%TYPE; 49 local2 table1.column2%TYPE; 50 51 52 BEGIN 53 54 SELECT column1, column2, proc1() 55 FROM table1; 56 57 proc2(); 58 59 local1 := local2; 60 61 DECLARE 62 63 64 65 66 67 68 69 70 71 72 73 74 END; 75 76 ----- 77 PROCEDURE proc2(param1 IN table1.column2 78 79 AS 80 81 local3 table1.column2%TYPE; 82 83 84 BEGIN 85 86 SELECT column1, column2 , proc1() 87 FROM table1; 88 proc2(proc3(proc1())); 89 90 END; 91 92 93 ----- 94 PROCEDURE proc2(</pre>
--	---

<p>Package_Reference</p> <p>References :</p> <ul style="list-style-type: none"> -> column1 -> column2 -> proc1 -> proc2 -> proc3 -> table1 -> l_varlocal1_inBlock -> l_varlocal2_inBlock -> local3 -> l_varlocal_1 -> local1 -> local2 -> param1 	<pre> 39 40 41 PACKAGE BODY Package_Reference 42 IS 43 44 ----- 45 PROCEDURE proc1 46 AS 47 48 local1 table1.column1%TYPE; 49 local2 table1.column2%TYPE; 50 51 52 BEGIN 53 54 SELECT column1, column2, proc1() 55 FROM table1; 56 57 proc2(); 58 59 local1 := local2; 60 61 DECLARE 62 63 local3 table1.column2%TYPE; 64 65 66 BEGIN 67 68 local1 := local2; 69 local1 := local1 + local2; 70 local3 := local1 + local3; 71 72 END; 73 74 END; 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 PROCEDURE proc3(95 param1 IN table1.column1% 96 param2 IN table1.column2% 97) 98 AS 99 100 l_varlocal_1 table1.column1%type; 101 102 BEGIN 103 104 SELECT column1, proc1() 105 FROM table1; 106 107 BEGIN 108 109 SELECT column1, column2, proc1() 110 FROM table1; 111 112 proc2(proc3(proc1())); 113 114 END ; 115 116 DECLARE 117 118 l_varlocal1_inBlock table1.column1 119 l_varlocal2_inBlock table1.column2 120 121 BEGIN 122 123 SELECT column1, column2, l_varloca 124 FROM table1; 125 126 proc2(proc3(proc1())); 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 </pre>
--	--

<p>Package_Reference</p> <p>References :</p> <ul style="list-style-type: none">-> column1-> column2-> proc1-> proc2-> proc3-> table1-> L_varlocal1_inBlock-> L_varlocal2_inBlock-> local3-> L_varlocal_1-> local1-> local2-> param1	<p>44</p> <p>45</p> <p>46</p> <p>47</p> <p>48</p> <p>49</p> <p>50</p> <p>51</p> <p>52</p> <p>53</p> <p>54</p> <p>55</p> <p>56</p> <p>57</p> <p>58</p> <p>59 ▶</p> <p>60</p> <p>61</p> <p>62</p> <p>63</p> <p>64</p> <p>65</p> <p>66</p> <p>67</p> <p>68 ▶</p> <p>69 ▶</p> <p>70</p> <p>71</p> <p>72</p> <p>73</p> <p>74</p> <p>75</p> <p>76</p> <p>77</p> <p>78</p> <p>79</p> <p>80</p>	<pre>----- PROCEDURE proc1 AS local1 table1.column1%TYPE; local2 table1.column2%TYPE; BEGIN SELECT column1, column2, proc1() FROM table1; proc2(); local1 := local2; DECLARE local3 table1.column2%TYPE; BEGIN local1 := local2; local1 := local1 + local2; local3 := local1 + local3; END; END; ----- PROCEDURE proc2(param1 IN table1.column2 AS I</pre>
--	---	--

The screenshot displays a code editor interface. On the left, a 'Package_Reference' pane shows a tree view of references including 'column1', 'column2', 'proc1', 'proc2', 'proc3', 'table1', and several local variables. The main editor area shows SQL code with line numbers 86 through 124. A red oval highlights a scroll bar at the bottom of the code view, with a list of line numbers (48, 54, 86, 95, 104, 109, 118, 123) visible below it, indicating references that are not currently visible in the code view.

When a source code contains several references, the line number of each reference is listed below the source code (see screenshot).

If you click on a line number, the view scrolls down to display the corresponding reference. This feature helps navigating through references in a large piece of code.

Some line numbers are highlighted in red: they are visible in the source code view. In the above example, there are 2 references not visible at line 48 and 54.

4.2.2. Stored Procedures

Visual Expert can list the items referenced by a stored procedure.

This is the same concept as the Oracle Packages, with a scope limited to a stored procedure:

The screenshot displays the Visual Expert interface. On the left, the 'Content' pane shows a tree view under 'Package_Reference' with a 'Definition' section containing 'proc1'. Below it, the 'References' section lists 'column1', 'column2', 'proc1', 'proc2', 'table1', 'local3', 'local1', and 'local2'. On the right, the code editor shows the SQL definition for 'proc1' and 'proc2'. The code for 'proc1' includes local variable declarations for 'column1' and 'column2', a 'SELECT' statement, and a call to 'proc2()'. The code for 'proc2' includes local variable declarations and arithmetic operations. Line numbers 42 through 80 are visible on the left side of the code editor. A status bar at the bottom shows '48 54'.

```
42 IS
43
44 -----
45 PROCEDURE proc1
46 AS
47
48     local1 table1.column1%TYPE;
49     local2 table1.column2%TYPE;
50
51
52 BEGIN
53
54     SELECT column1, column2, proc1()
55     FROM table1;
56
57     proc2();
58
59     local1 := local2;
60
61 DECLARE
62
63     local3 table1.column2%TYPE;
64
65
66 BEGIN
67
68     local1 := local2;
69     local1 := local1 + local2;
70     local3 := local1 + local3;
71
72 END;
73
74 END;
75
76 -----
77 PROCEDURE proc2(param1 IN table1.column2
78 AS
79
80
```

4.2.3. Other DB Code items

Visual Expert also analyses Views, Types and Cursors.

It supports references to tables and columns with on the instruction %Type

4.2.4. PL/SQL %TYPE references

Visual Expert supports all %TYPE references. Parent items (Views, Types, and Cursors) and the table/column referenced. For instance:

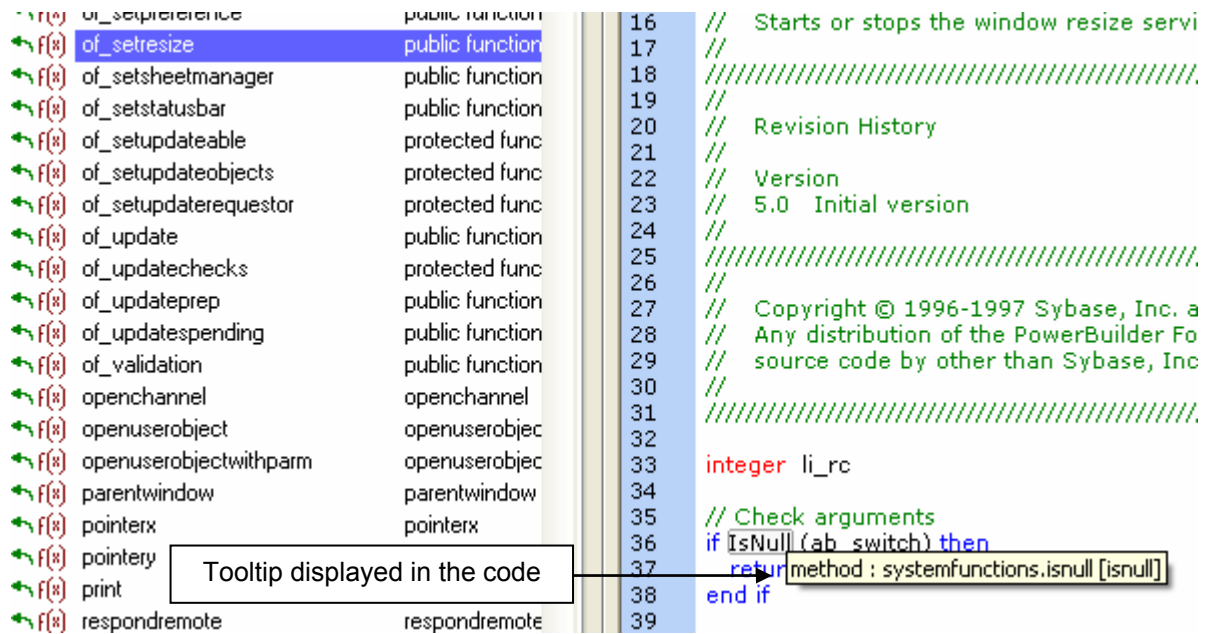
Content : Package_Reference Definition : S:\VELOG\2008-11-07 proc1 References : -> column1 -> column2 -> proc1 -> proc2 -> table1 -> local3 -> local1 -> local2 proc2 proc3	44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66	<pre>PROCEDURE proc1 AS local1 table1.column1%TYPE; local2 table1.column2%TYPE; BEGIN SELECT column1, column2, proc1() FROM table1; proc2(); local1 := local2; DECLARE local3 table1.column2%TYPE; BEGIN</pre>
Content : Package_Reference Definition : S:\VELOG\2008-11-07 proc1 proc2 proc3 References : -> column1 -> column2 -> proc1 -> proc2 -> proc3 -> table1 -> l_varlocal1_inBloc -> l_varlocal2_inBloc -> l_varlocal_1	93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114	<pre>PROCEDURE proc3(param1 IN table1.column1%TYPE, param2 IN table1.column2%TYPE) AS l_varlocal_1 table1.column1%type; BEGIN SELECT column1, proc1() FROM table1; BEGIN SELECT column1, column2, proc1() FROM table1; proc2(proc3(proc1())); END ;</pre>

4.2.5. Parameters & Local Variables

Variables & Parameters are referenced as any other DB Item. Each reference will be highlighted in the source code.

5. Tooltips in the code

When you move the mouse over the source code, ToolTips are now displayed. They provide additional information about an item referenced in the code (object, method or variable).



The screenshot shows a code editor with a list of methods on the left and source code on the right. A tooltip is displayed over the 'isNull' method call in the code. The tooltip contains the text 'Tooltip displayed in the code' and an arrow pointing to the 'isNull' method call in the code. The code on the right is as follows:

```
16 // Starts or stops the window resize servi
17 //
18 ////////////////////////////////////////////////////////////////////
19 //
20 // Revision History
21 //
22 // Version
23 // 5.0 Initial version
24 //
25 ////////////////////////////////////////////////////////////////////
26 //
27 // Copyright © 1996-1997 Sybase, Inc. a
28 // Any distribution of the PowerBuilder Fo
29 // source code by other than Sybase, Inc
30 //
31 ////////////////////////////////////////////////////////////////////
32
33 integer li_rc
34
35 // Check arguments
36 if IsNull (ab_switch) then
37     return method : systemfunctions.isnull [isnull]
38 end if
39
```

5.1. PowerBuilder objects

A tooltip will display detailed information about a PB Object:

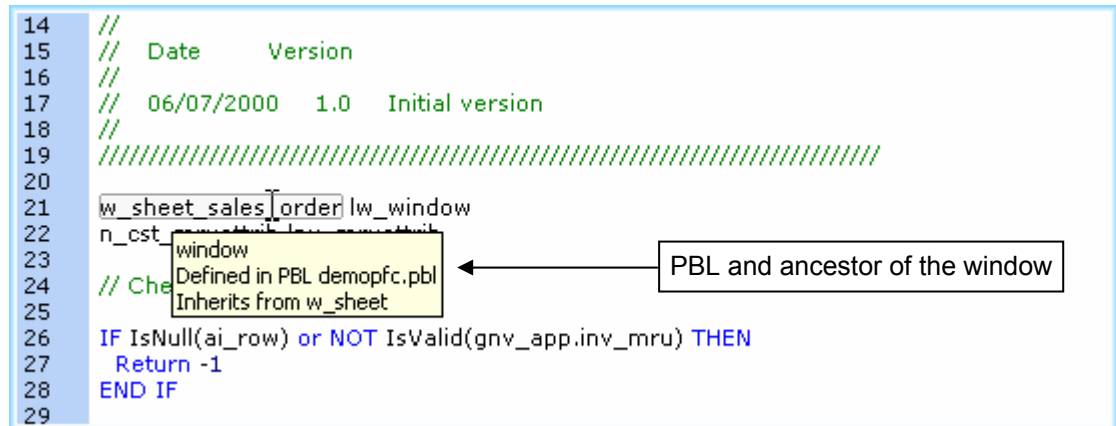
- Its type (window, userobject, menu ...)
- Its ancestor
- The PBL containing this object

Examples :

```

14 //
15 // Date      Version
16 //
17 // 06/07/2000  1.0  Initial version
18 //
19 ///////////////////////////////////////////////////////////////////
20
21 w_sheet_sales_order lw_window
22 n_cst
23 // Che
24 // Inherits from w_sheet
25
26 IF IsNull(ai_row) or NOT IsValid(gnv_app.inv_mru) THEN
27     Return -1
28 END IF
29

```

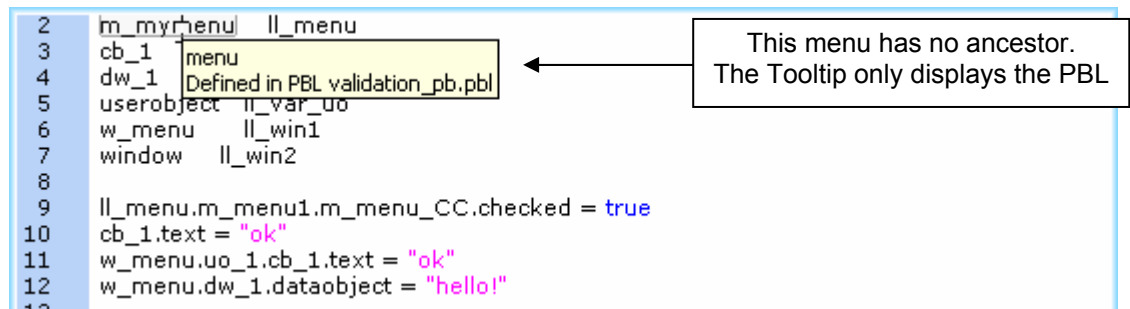


PBL and ancestor of the window

```

2  m_myrmenu ll_menu
3  cb_1      menu
4  dw_1      Defined in PBL validation_pb.pbl
5  userobject ll_var_uo
6  w_menu    ll_win1
7  window    ll_win2
8
9  ll_menu.m_menu1.m_menu_CC.checked = true
10 cb_1.text = "ok"
11 w_menu.uo_1.cb_1.text = "ok"
12 w_menu.dw_1.dataobject = "hello!"
13

```



This menu has no ancestor.
The Tooltip only displays the PBL

5.2. PowerBuilder Controls

```

2  m_mymenu ll_menu
3  cb_1     ll_var_cb
4  dw_1     ll_var_dw
5  userobject ll_var_uo
6  w_menu   ll_win1
7  window   ll_win2
8
9  ll_menu.m_menu1.m_menu_CC.checked = true
10 cb_1.text = "ok"
11 w_menu.uo_1.cb_1.text = "ok"
12 w_menu.dw_1.dataobject = m_menu
13
14 uo_1.cb_1
15 dw_1.dataobject = m_menu
16
17 m_menu.text="ok"

```

Control's ancestor and PBL

userobject
Defined in PBL validation_pb.pbl
Inherits from u_test_view1

```

2  m_mymenu ll_menu
3  cb_1     ll_var_cb
4  dw_1     ll_var_dw
5  userobject ll_var_uo
6  w_menu   ll_win1
7  window   ll_win2
8
9  ll_menu.m_menu1.m_menu_CC.checked = true
10 cb_1.text = "ok"
11 w_menu.uo_1.cb_1.text = "ok"
12 w_menu.dw_1.dataobject = m_menu
13
14 uo_1.cb_1
15 dw_1.dataobject = m_menu
16
17 m_menu.text="ok"

```

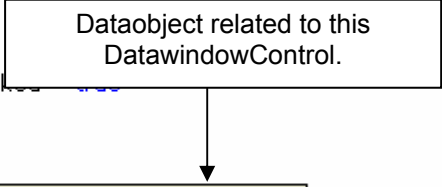
Same for nested controls

commandbutton
Control of w_menu.uo_1
Inherits from u_test_view1.cb_1

5.3. Datawindows

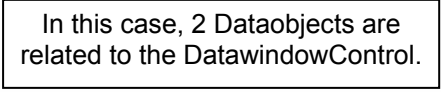
All dataobjects associated to a DatawindowControl are listed in the Tooltip (both dataobjects associated with the PB painter and dynamically associated in code):

```
2 m_mymenu ll_menu
3 cb_1 ll_var_cb
4 dw_1 ll_var_dw
5 userobject ll_var_uo
6 w_menu ll_win1
7 window ll_win2
8
9 ll_menu.m_menu1.m_menu_CC.checked = true
10 cb_1.text = "ok"
11 w_menu.uo_1.cb_1.text = "ok"
12 w_menu.dw_1.dataobject = "hello!"
13
14 uo_1.cb_1.text = "ok"
15 dw_1.dataobject = "hello!"
16
17 m_menu.text="ok"
```



Dataobject related to this DatawindowControl.

```
2 m_mymenu ll_menu
3 cb_1 ll_var_cb
4 dw_1 ll_var_dw
5 userobject ll_var_uo
6 w_menu ll_win1
7 window ll_win2
8
9 ll_menu.m_menu1.m_menu_CC.checked = true
10 cb_1.text = "ok"
11 w_menu.uo_1.cb_1.text = "ok"
12 w_menu.dw_1.dataobject = "hello!"
13
14 uo_1.cb_1.text = "ok"
15 dw_1.dataobject = "hello!"
16
17 m_menu.text="ok"
```



In this case, 2 Dataobjects are related to the DatawindowControl.

5.4. PowerBuilder variables

Tooltip will display the scope and type of each variable, as well as detailed information about this type:

```

2  m_myMENU  ll_menu
3  cb_1      menu
4  dw_1      Defined in PBL validation_pb.pbl
5  userobject ll_var_uo
6  w_menu    ll_win1
7  window    ll_win2
8
9  ll_menu.m_menu1.m_menu_CC.checked = true

```

This tooltip displays information about the menu "m_myMENU".

```

2  m_myMENU  ll_menu
3  cb_1      ll_var_cb
4  dw_1      ll_var_dw
5  userobject ll_var_uo
6  w_menu    ll_win1
7  window    ll_win2
8
9  ll_menu.m_menu1.m_menu_CC.checked =
10 cb_1
11 w_menu
12 w_menu
13
14 uo_1
15 dw_1.dataobject = "hello!"
16

```

This one displays information on the **scope** (local), and the **type** (menu "m_myMENU") of the variable ll_menu

```

2  m_myMENU  ll_menu
3  cb_1      ll_var_cb
4  dw_1      ll_var_dw
5  userobject ll_var_uo
6  w_menu    ll_win1
7  window    ll_win2
8
9  ll_menu.m_menu1.m_menu_CC.checked =
10 cb_1
11 w_menu
12 w_menu
13
14 uo_1
15 dw_1.dataobject = "hello!"
16

```

The type of this variable is a DatawindowControl.
The Tooltip displays the scope of the variable (local), its type ('dw_1'), and the dataobjects related to 'dw_1'.

5.5. PowerBuilder methods

Functions tooltips will indicate :

- The prototype and owner of a function
- « Built-in function » if this is a PowerBuilder built-in function

Examples :

```
24 // Check parameters.
25
26 IF IsNull(ai_row) or NOT IsValid(gnv_app.inv_mru) THEN
27     Return -1
28 END IF
29
30 // Retrieve row from DataStore.
31
32 gnv_app.inv_mru.of_GetItem (ai_row, Inv_mruattrib)
33
34 message.of_setdoubleparm
35
```

public function integer of_getitem(long, ref n_cst_mruattrib)
Belong to pfc_n_cst_mru

```
31
32 gnv_app.inv_mru.of_GetItem (ai_row, Inv_mruattrib)
33
34 message.of_setdoubleparm (long(Inv_mruattrib.is_menuitemkey))
35
36 OpenSheet(lw_window, Inv_mruattrib.is_classname, This.parentwindow(), 0,Original!)
37
38 Return 1
39
```

Built-in function of window

```
11
12 int i_pos_start, i_pos_end, i_pos_start_connectstring, i_pos_end_connectstring
13 string s_dbparm, s_insert
14
15 CHOOSE CASE vg_f_get_dbms(atr_trans)
16
17 CASE "sybase", "mdia" function_object
18     atr_trans.logid = trim(as_code)
19     atr_trans.logpass = trim(as_password)
20 CASE "informix"
21     atr_trans.userid = trim(as_code)
22     atr_trans.dbpass = trim(as_password)
23 CASE "odbc"
24     atr_trans.userid = trim(as_code)
25     atr_trans.dbpass = trim(as_password)
```

Defined in PBL vg_security.pbl

5.6. Oracle, T-SQL and Database objects

At this point (VE6.0 beta3), just a few tooltips are available for Oracle, T-SQL and Database items.

This feature will be extended in future releases.

For example:

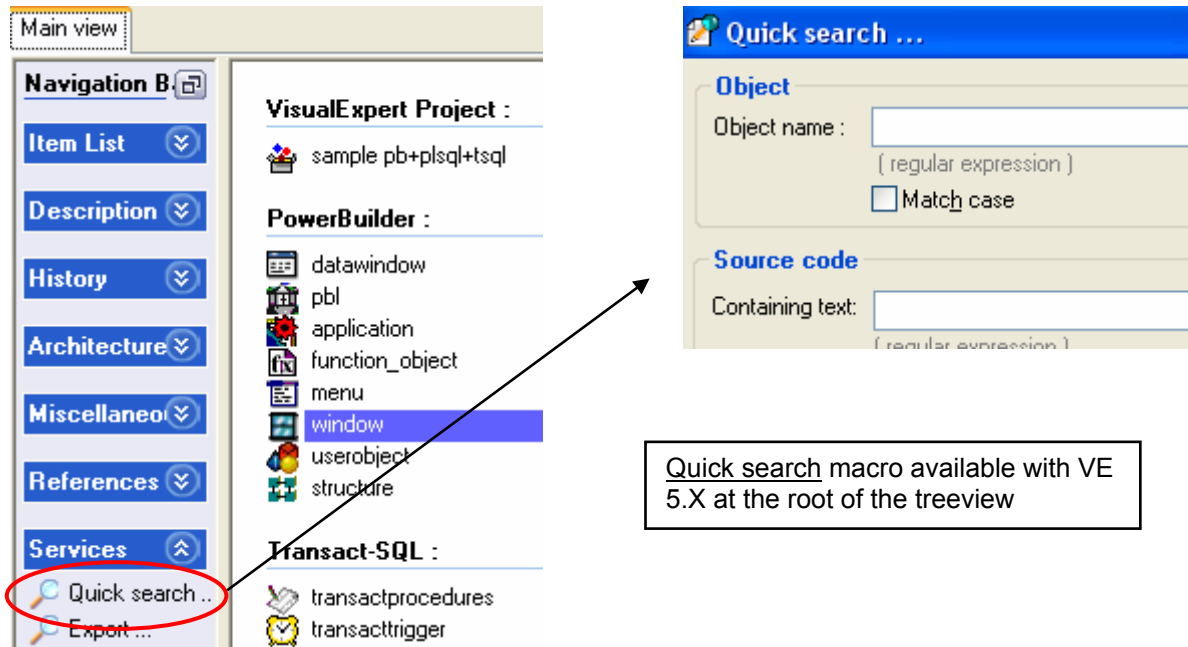
```
138
139     begin
140
141     select fname, lname
142         into Cust_fname, Cust_lname
143     from customer
144     where cust_id = Customer_Id ;
145     SELECT Count(*) into Prod_num
146     FROM customer,
147         product,
148         sales order items.
```

```
153         ( sales_order.id = sales_order_items.id )and
154         (product.id = sales_order_items.prod_id)
155     GROUP BY product.id;
156
157
158     dbms_output.put_line('-----')
159     dbms_output.put_line(
160     'Product list ' || uf_SetBracket(tochar(Prod_num)) ||
161     uf_GetFullname(Cust_fname) Stored procedure 'uf_SetBracket'
162     to_char(sysdate, 'dd-Mon hh24:mi:ss')
163     );
164     dbms_output.put_line('-----')
```

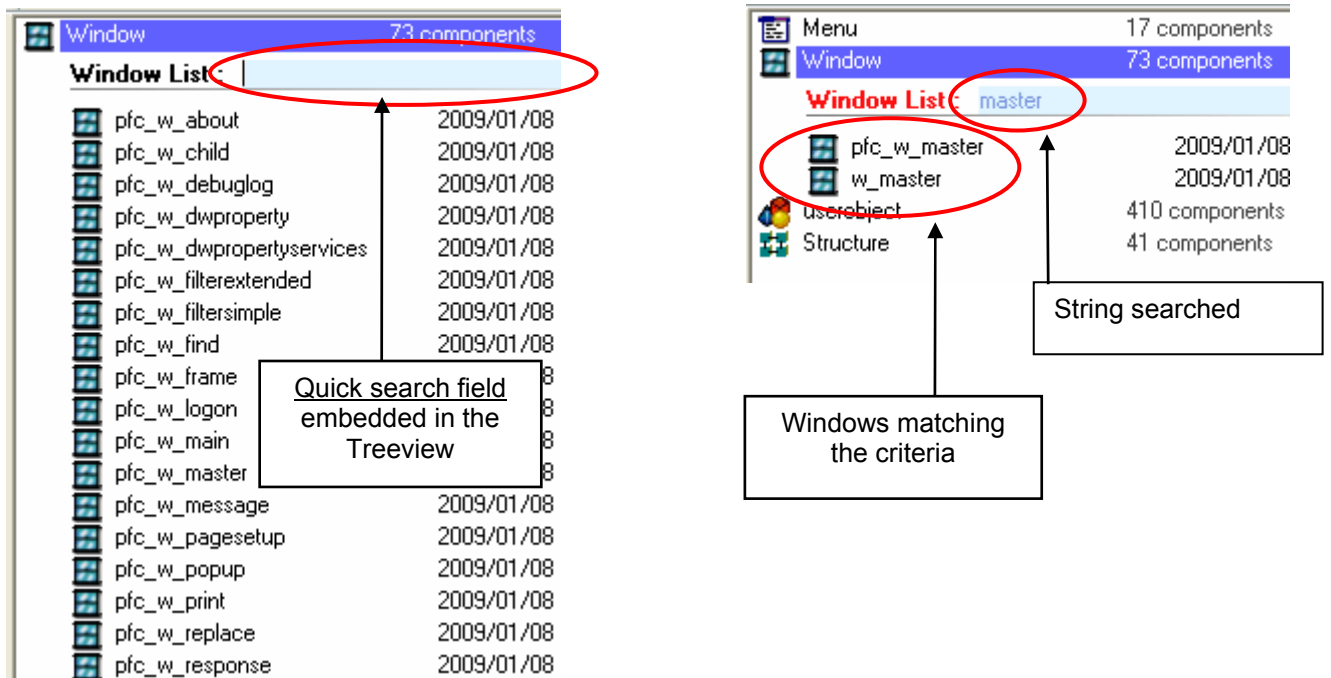
6. New Search Feature in the Treeview

6.1. Quick search in the treeview

Visual Expert 5.x offered a “Quick Search” macro.
This macro could only search at the root of the treeview.



Visual Expert 6.0 replaces this feature with a quick search field, available at every level / for every node of the treeview:



In VE6.0 beta2, only a flat list of components is supported (no hierarchy or sub-results).
This feature will be improved in the official release. Again, any comments are welcome...

7. Miscellaneous

- Polymorphism is now supported with exceptions declared in VE project
- Visual Expert GUI has been migrated to PB11 for future integration of .NET controls in VE
- PB11 Pre-processing supported: C# code is identified and ignored for now.