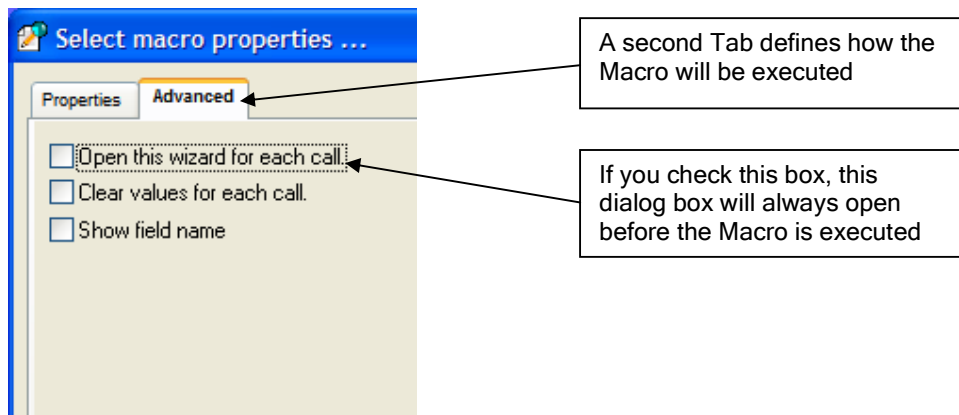
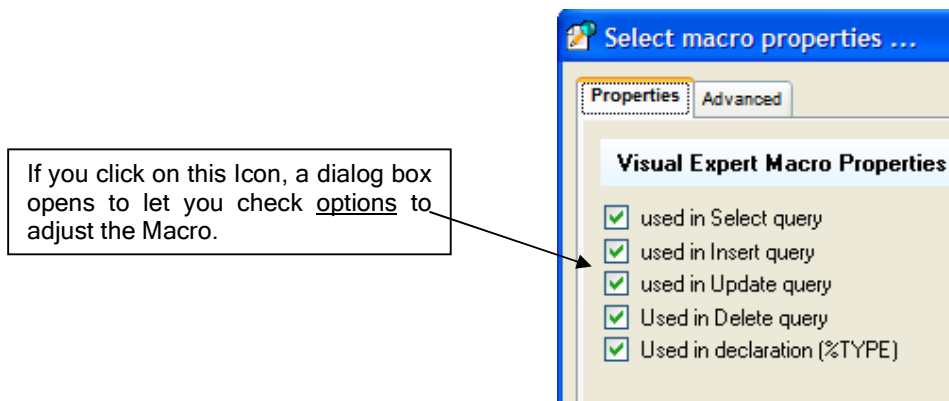
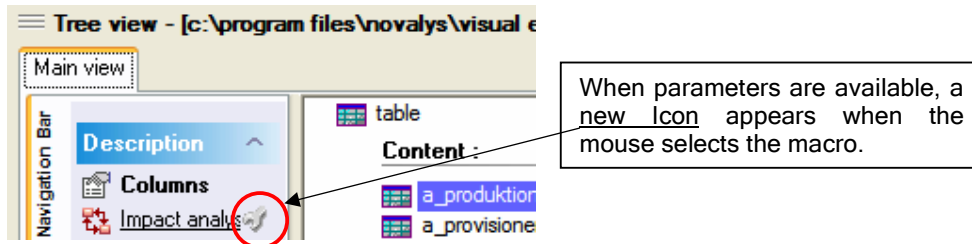


# Introduction to Visual Expert 6.0 new features

<b>1. NEW NAVIGATION BAR.....</b>	<b>2</b>
<b>2. NEW MACROS.....</b>	<b>3</b>
2.1. LESS MACROS, MORE PARAMETERS.....	3
2.2. MACROS SHORTCUTS IN THE TREEVIEW MENU .....	3
2.3. NEW MACROS TO LIST OBJECTS IN THE TREEVIEW.....	4
2.4. NEW MACROS TO IDENTIFY SPECIFIC DATAWINDOWS.....	4
2.5. MAPPING DATAWINDOWS / DBMS.....	6
2.6. POWERBUILDER ⇔ STORED PROCEDURES DEPENDENCIES .....	6
2.7. INHERITANCE HIERARCHY (WINDOWS/UO/MENUS).....	7
<b>3. NEW IMPACT ANALYSIS .....</b>	<b>8</b>
3.1. PRESENTATION OF THE RESULT.....	8
3.1.1. <i>In the treeview</i> .....	8
3.1.2. <i>In the source code view</i> .....	9
3.2. IMPACT ANALYSIS EXAMPLES .....	10
3.2.1. <i>Impact Analysis on a Table</i> .....	10
3.2.2. <i>Impact Analysis on a Database Column</i> .....	14
3.2.3. <i>Impact Analysis on a Stored Procedure</i> .....	17
<b>4. EXPLORING REFERENCES WITH VISUAL EXPERT 6.0 .....</b>	<b>19</b>
4.1. REFERENCES FROM PB TO THE DBMS .....	19
4.1.1. <i>Datwindow DataSource</i> .....	19
4.1.2. <i>Tables and Columns referenced by a DataWindow DataSource</i> .....	21
4.1.3. <i>Procedures referenced by a DataWindow DataSource</i> .....	22
4.1.4. <i>Tables and Columns referenced by a PowerScript</i> .....	23
4.1.5. <i>Other PowerBuilder References</i> .....	27
4.1.5.1. <i>Cursors and Procedures declared as members of a Component</i> .....	27
4.1.5.2. <i>Cursors and Procedures declared in a PowerScript</i> .....	30
4.1.5.3. <i>RPC FUNC</i> .....	32
4.2. REFERENCES BETWEEN PL/SQL AND T-SQL COMPONENTS .....	34
4.2.1. <i>Oracle Package</i> .....	35
4.2.2. <i>Stored Procedures</i> .....	41
4.2.3. <i>Other DB Code items</i> .....	42
4.2.4. <i>PL/SQL %TYPE references</i> .....	42
4.2.5. <i>Parameters &amp; Local Variables</i> .....	42
<b>5. NEW SOURCE CODE VIEW.....</b>	<b>43</b>
5.1. HYPERLINKS IN THE CODE .....	43
5.2. TITLE OF THE SOURCE CODE VIEW .....	43
5.3. SEARCH IN THE SOURCE CODE VIEW .....	43
5.4. TOOLTIPS IN THE CODE .....	44
5.4.1. <i>PowerBuilder objects</i> .....	45
5.4.2. <i>PowerBuilder Controls</i> .....	46
5.4.3. <i>Datawindows</i> .....	47
5.4.4. <i>PowerBuilder variables</i> .....	48
5.4.5. <i>PowerBuilder methods</i> .....	49
5.4.6. <i>Oracle, T-SQL and Database objects</i> .....	50
<b>6. MISCELLANEOUS .....</b>	<b>51</b>
6.1. TECHNICAL REQUIREMENTS .....	51
6.2. MANAGE A SHORT-LIST OF COMPONENTS .....	51

# 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.



## 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

**Component list for PowerBuilder :**

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 :**

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

**Transact SQL :**

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

**Stored proc. list :**

- OrdersbyEmployee
- Productsbycustomer
- sp\_customer\_list
- sp\_deleteemployee

**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

### 2.2. Macros shortcuts in the Treeview Menu

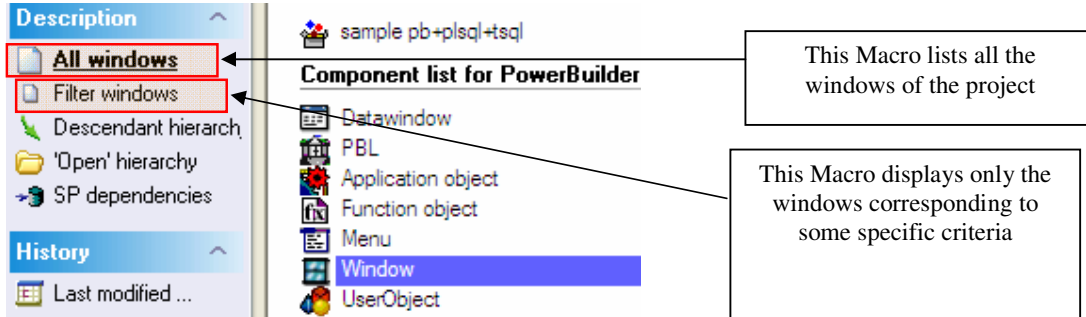
The pop-menu in the treeview (right-click on an item in the treeview) now includes the main macros available for the selected item – no need to go to the navigation bar.

**Main macros available in the treeview Pop-menu**

## 2.3. New Macros to list objects in the treeview

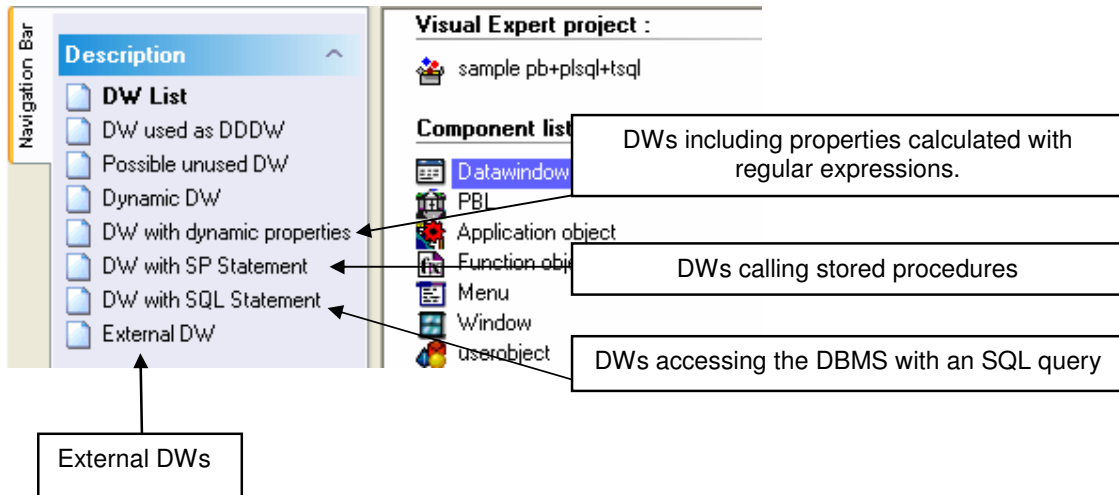
Objects can now be listed in the treeview with 2 macros:

- The macro “**All <Object>**” lists all the objects of this type.  
This macro is executed by default with a double-click at the root of the treeview.
- The macro “**Filter <Object>**” offers various parameters to filter each type of object.



## 2.4. New macros to identify specific DataWindows

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



**DW with SQL Statement :**

C:\Program Files\Novals\Visual Expert 6.0\Sample\sample.pb\\*.pbl

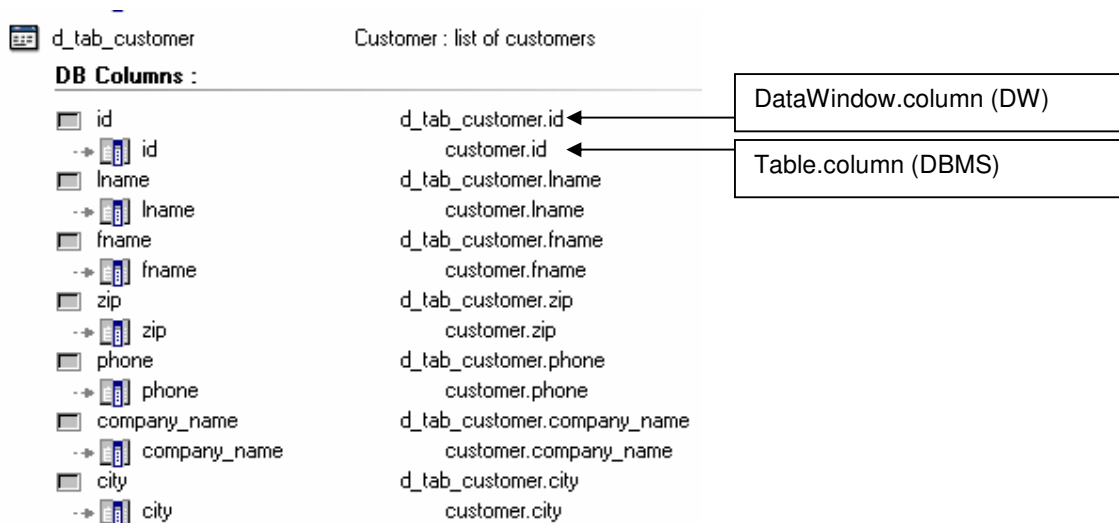
- demopfc.pbl
  - d\_ff\_employee\_general
  - d\_ff\_employee\_personal
  - d\_ff\_employee\_professional
  - d\_ff\_sales\_order
  - d\_tab\_customer
  - d\_tab\_department
  - d\_tab\_employee
  - d\_tab\_fin\_cod
  - d\_tab\_product
  - d\_tab\_product\_view2
  - d\_tab\_product\_view3
  - d\_tab\_sales\_order
  - d\_tab\_sales\_order\_items
  - d\_tab\_salesperson
  - dddw\_product
  - dddw\_product\_view2
  - dddw\_product\_view3
- pfcapsrv.pbl
  - d\_definedmessages
  - d\_pfcsecurity\_allcontrollist

Each DW Macro now generates a result grouped by PBL, making it easier to browse.

## 2.5. Mapping DataWindows / DBMS

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

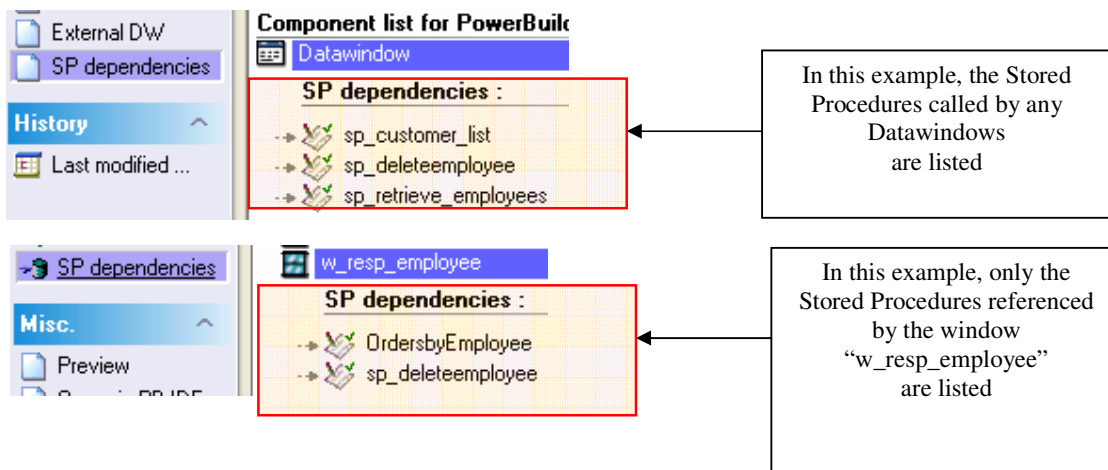
This macro displays the DW- DBMS mapping: it shows which table.column in the DBMS corresponds to each DW column.



## 2.6. PowerBuilder ↔ Stored Procedures dependencies

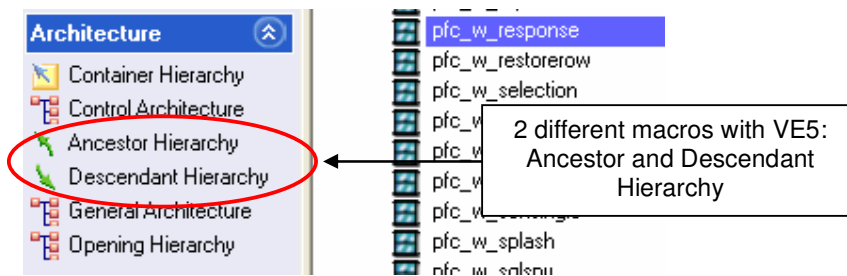
A new Macro called « SP dependencies » lists all stored procedures referenced by PB objects:

- This macro is available at the root of the treeview for each type of PB components
- It is also available for PB objects displayed in the treeview

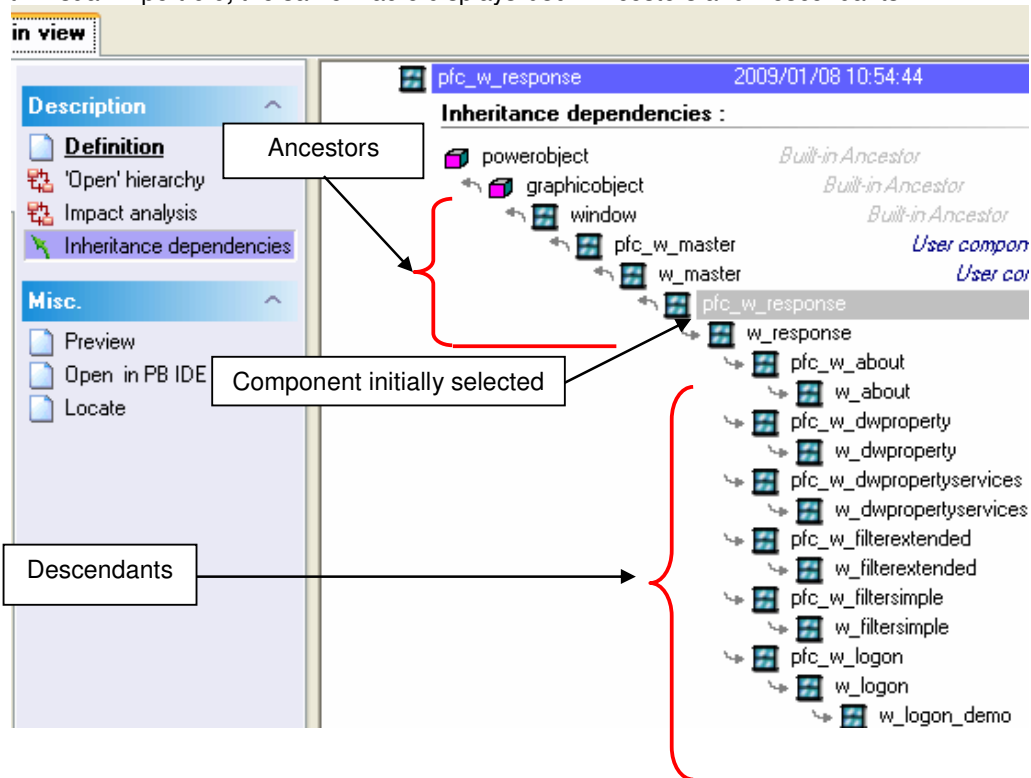


## 2.7. Inheritance hierarchy (Windows/UO/Menus)

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 simpler use.

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

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 to Bonus)
  - ue\_sql\_test (pin icon)
  - bonus\_cursor (pin icon, dependency symbol to Bonus)
  - ue\_declare (pin icon)
  - bonus\_cursor (pin icon, dependency symbol to Bonus)
- d\_proc3
- data source
- d\_bonus
- data source
- S:\WELOG\2008-11-07 15-57-44 [MITERRA] [WILLIAM SIGUE
- Test\_Package
  - proc\_SQL (pin icon)
    - {BEGIN SELECT a,b,c,d FROM tt; 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 before, 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 consists of two screenshots of the Visual Expert interface, illustrating the results of an Impact Analysis for a PL/SQL Procedure « Proc 1 ».

**Top Screenshot:** Shows the source code view for a package. The treeview on the left highlights a file containing two packages. A callout box explains: "This example shows an Impact Analysis for the PL/SQL 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 shows a procedure with a call to 'proc3'. Below the code, a list of line numbers (54, 86, 88, 104, 109, 112, 126, 128, 132, 153, 156) is displayed, with an arrow pointing to line 112.

**Bottom Screenshot:** Shows the source code view for a specific PL/SQL block. A callout box explains: "If we select a PL/SQL Block in the treeview, only 2 references included in this block are listed." The source code shows a 'SELECT' statement that references 'column1' and 'column2'. Below the code, a list of line numbers (109, 112) is displayed, 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 »

The image displays two screenshots from SQL Developer illustrating impact analysis on a table named 'bonus'.

**Top Screenshot:** Shows the 'bonus' table selected in the tree view. The impact analysis code is displayed in the editor, starting with line 45:

```
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
```

**Bottom Screenshot:** Shows the 'ue\_sql\_test()' event selected in the tree view. The impact analysis code is displayed in the editor, starting with line 1:

```
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
```

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'bonus' folder is expanded under 'Impact analysis :'. The tree structure includes:

- validation\_pb
  - w\_sql
    - ue\_sql
    - ue\_sql\_test
    - bonus\_cursor (with SQL icon)
      - ue\_declare
      - bonus\_cursor (with SQL icon)
  - d\_proc3
    - data source (highlighted)
  - d\_bonus
    - data source
- S:\VELOG\2008-11-07 15-57-4
  - Test\_Package
    - proc\_SQL
      - {BEGIN SELEC

On the right, the 'Dwdatasource - d\_proc3.data source inherited from dwdata' window shows a query editor with the following SQL code:

```
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:\VELOG\2008-11-07 15-57-4
    - Test\_Package
      - proc\_SQL

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:\VELOG\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"

The screenshot displays the Visual Expert 6.0 interface for impact analysis. The left pane shows the database structure for 'bonus', with the 'ue\_sql\_test' object selected under the 'w\_sql' folder. The right pane shows the SQL code for the event 'event ue\_sql\_test() from w\_sql', which includes a select statement for 'bonus\_date' and a cursor declaration for 'bonus\_cursor'.

**Event - event ue\_sql\_test() from w\_sql**

```

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

```

**Logicalcursorname - bonus\_cursor inherited**

```

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,employee
20 into :ls1, :ls2
21 from employee;
22
23

```

**bonus**

**Columns :**

- bonus\_amount
- bonus\_date

**Impact analysis :**

- validation\_pb
  - w\_sql
    - ue\_sql
    - ue\_sql\_test
    - bonus\_cursor
      - ue\_declare
        - bonus\_cursor
- d\_proc3
  - data source
  - bonus\_date
- d\_bonus
  - data source
  - bonus\_date

S:\WELOG\2008-11-07 15-57-44 [v]

- Test\_Package
  - proc\_SQL

{BEGIN SELECT a,}

**Dwdatasource - d\_proc3.data source inherit**

Find

```

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

```

---

**bonus**

**Columns :**

- bonus\_amount
- bonus\_date

**Impact analysis :**

- validation\_pb
  - w\_sql
    - ue\_sql
    - ue\_sql\_test
    - bonus\_cursor
      - ue\_declare
        - bonus\_cursor
  - d\_proc3
    - data source
    - bonus\_date
  - d\_bonus
    - data source
    - bonus\_date

S:\WELOG\2008-11-07 15-57-44 [v]

  - Test\_Package
    - proc\_SQL

{BEGIN SELECT a,}

```

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

```

Impact Analysis on the column « table1.column1 »

<ul style="list-style-type: none"> <li>table1</li> <li>Columns :</li> <li>column1</li> <li>Impact analysis :</li> <li>S:\WELOG\2008-11</li> <li>Package_Refe             <ul style="list-style-type: none"> <li>proc3</li> <li>{BEC}</li> <li>{DEC}</li> <li>proc1</li> <li>proc2</li> <li>Package_Refe                 <ul style="list-style-type: none"> <li>proc4</li> </ul> </li> </ul> </li> <li>S:\WELOG\2008-11</li> <li>Test_Package             <ul style="list-style-type: none"> <li>csie_invoi                 <ul style="list-style-type: none"> <li>{BEC}</li> <li>{DEC}</li> </ul> </li> <li>block_ref                 <ul style="list-style-type: none"> <li>{BEC}</li> <li>{BEC}</li> </ul> </li> </ul> </li> <li>column2</li> </ul>	<pre> 93 94 95 ▶ 96 97 98 99 100 ▶ 101 102 103 104 ▶ 105 106 107 108 109 ▶ 110 111 112 113 114 115 116 117 118 ▶ 119 120 121 122 123 ▶ 124 125 126 127 128 129 130 131 132 133 134 135 </pre>	<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 ;     DECLARE         l_varlocal1_inBlock table1.column1%typ         l_varlocal2_inBlock table1.column2%typ     BEGIN         SELECT column1, column2, l_varlocal_1         FROM table1;         proc2( proc3( proc1() ) );         l_varlocal1_inBlock := proc3(l_varloc     END ;     proc2( proc1() ); END ; </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      Proce  204
              205      BEGIN
              206
              207      SELECT
              208          pref_database.pref_user.pr
              209          col2,
              210          col3 + col4 + col5,
              211          f(col10,col11 + 10 + xxxxxx
              212
              213      FROM table1, table2 as x
              214      ;
              215      proc1( proc2() );
              216
              217      BEGIN
              218
              219          BEGIN
              220
              221              SELECT column1, column2
              222              FROM table1;
              223
              224          proc2( proc3( proc1() ) );
              225
              226      END ;
              227
              228      BEGIN
              229
              230          SELECT column1, column2
              231          FROM table1;
              232
              233          proc2( proc3( proc1() ) );
              234
              235      END ;
              236
              237      SELECT column1
              238      FROM table1;
              239
              240      END;
              241
              242      END ;
```

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

The screenshot displays the SQL Developer interface during an impact analysis. On the left, a tree view shows a project structure under 'proc1'. The 'Impact analysis' pane is active, showing a tree of objects including 'validation\_pb', 'd\_procstock\_2', 'data source', 'Package\_Reference\_2', 'proc4', 'Test\_Package', 'csie\_invoice\_summary\_B', 'proc\_SQL', 'TEST1', and 'block\_ref\_sample'. The 'proc1' object is selected, and its internal structure is visible. On the right, the PL/SQL code for 'proc1' is shown, spanning lines 204 to 242. A red box highlights a nested block call to 'proc1()' within the 'proc2' procedure. Two red arrows point from the tree view to the highlighted code: one from the 'proc1' object and another from a sub-object within the 'Test\_Package'.

```

204
205 BEGIN
206
207     SELECT
208         pref_database.pref_user.pr
209         col2,
210         col3 + col4 + col5,
211         f(col10,col11 + 10 + xxxxx
212
213     FROM table1, table2 as x
214     ;
215     proc1( proc2() );
216
217 BEGIN
218
219     BEGIN
220
221         SELECT column1, column2
222         FROM table1;
223
224         proc2( proc3( proc1() ) );
225
226     END ;
227
228 BEGIN
229
230     SELECT column1, column2
231     FROM table1;
232
233     proc2( proc3( proc1() ) );
234
235     END ;
236
237     SELECT column1
238     FROM table1;
239
240 END;
241
242 END ;

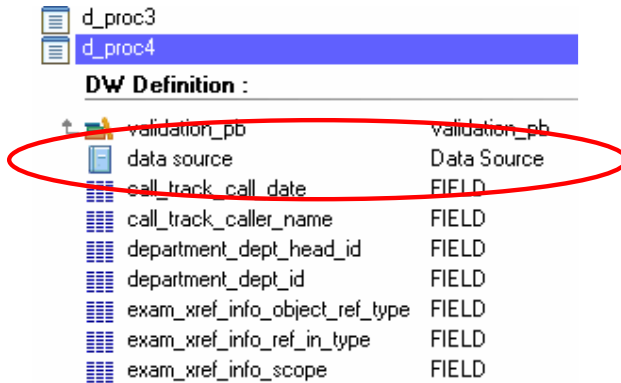
```

## 4. Exploring references with Visual Expert 6.0

### 4.1. References from PB to the DBMS

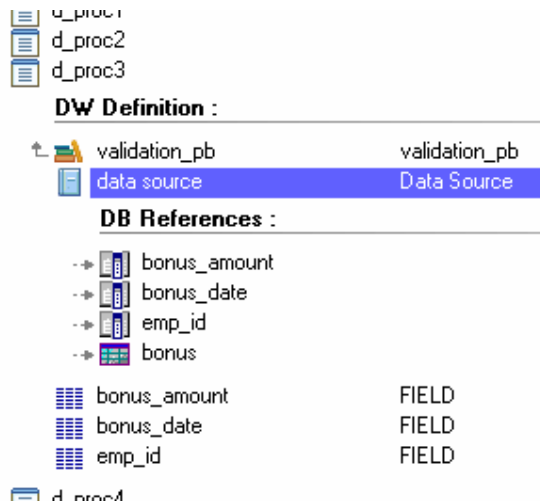
#### 4.1.1. Datawindow DataSource

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

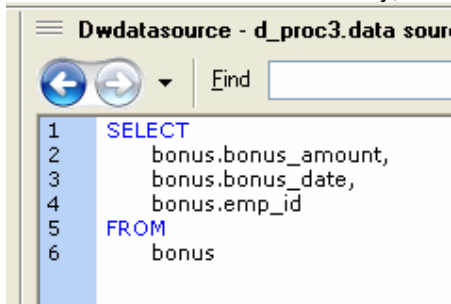


A "DataSource" is displayed when the DW has a data source (SQL Query or Stored Procedure).

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:

d\_html\_datawindow  
d\_proc1

**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 up
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:

The image displays three sequential screenshots of a software interface for defining a DataWindow DataSource. Each screenshot shows a treeview on the left and a SQL editor on the right.

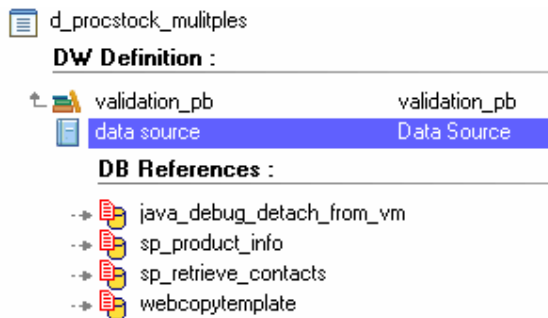
- Top Screenshot:** The treeview shows 'd\_proc3' expanded to 'validation\_pb' > 'data source'. Under 'DB References', 'bonus\_amount' is selected. The SQL editor shows a query: 

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

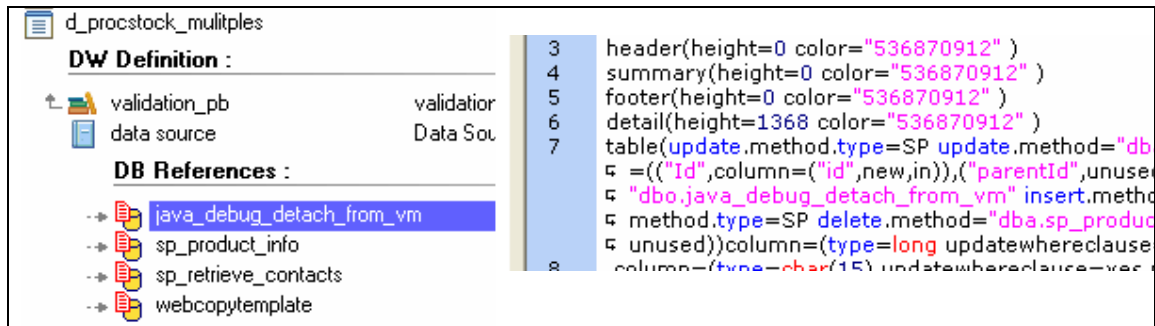
 The text 'bonus.bonus\_amount' on line 2 is highlighted in blue.
- Middle Screenshot:** The treeview shows 'bonus\_date' selected. The SQL editor shows the same query, but now 'bonus.bonus\_date' on line 3 is highlighted in blue.
- Bottom Screenshot:** The treeview shows 'bonus' selected. The SQL editor shows the same query, but now 'bonus' on line 6 is highlighted in blue.

### 4.1.3. Procedures referenced by a DataWindow DataSource

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



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



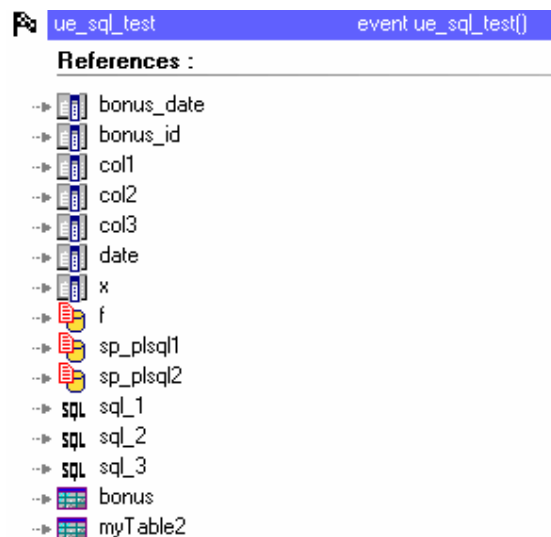
#### 4.1.4. Tables and Columns referenced by a PowerScript

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

```

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_plsc
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

```

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_plsc
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

```

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_plsq
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 :**

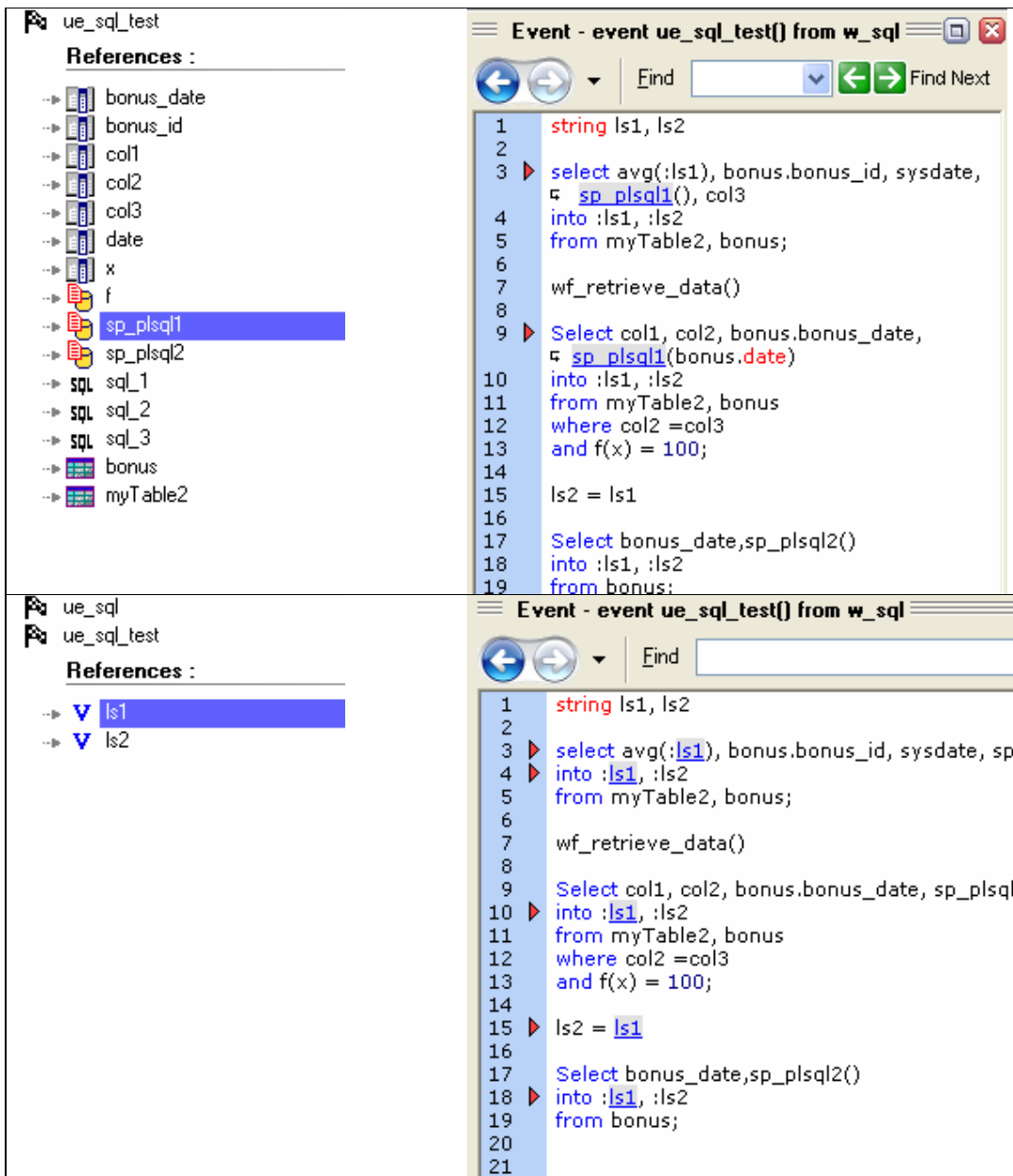
- > bonus\_date
- > bonus\_id
- > col1
- > col2
- > col3
- > date
- > x
- > f
- > sp\_plsql1
- > sp\_plsql2
- > SQL sql\_1
- > SQL sql\_2
- > SQL sql\_3
- > bonus
- > myTable2

Event - event ue\_sql\_test() from w\_sql

```

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_plsq
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.

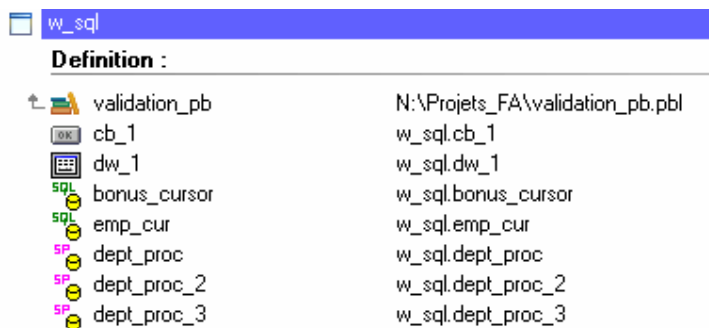
3 possibilities:

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:



The screenshot shows a window definition table for a component named 'w\_sql'. The table lists various members and their corresponding paths. The members include a validation component, a control, a data window, two cursors, and three procedures.

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:

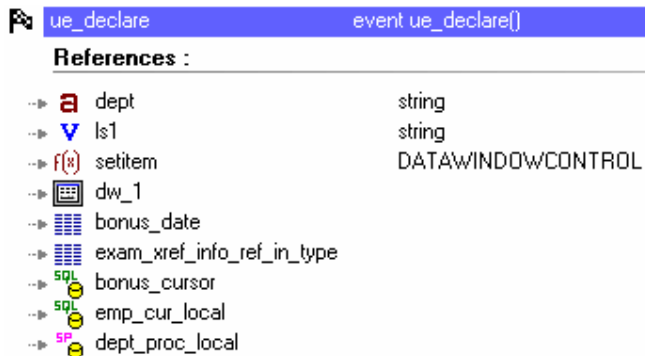
<p>w_sql</p> <p><b>Definition :</b></p> <ul style="list-style-type: none"> <li>validation_pb</li> <li>cb_1</li> <li>dw_1</li> <li>bonus_cursor</li> <li>emp_cur</li> <li><b>dept_proc</b></li> <li>dept_proc_2</li> <li>dept_proc_3</li> <li>dept</li> <li>i_member2</li> <li>s_lastmember</li> <li>s_lastmember2</li> <li>s_lastmember3</li> </ul>	<p>26</p> <p>27</p> <p>28</p> <p>29</p> <p>30</p> <p>31</p> <p>32</p> <p>33</p> <p>34</p> <p>35</p> <p>36</p> <p>37</p> <p>38</p> <p>39</p> <p>40</p> <p>41</p> <p>42</p> <p>43</p> <p>44</p>	<pre>dw_1 dw_1 cb_1 cb_1 end type global w_sql w_sql  type variables  // string dept ▶ DECLARE dept_proc PROCEDURE FOR pref_user.spm1(:dept); DECLARE dept_proc_2 PROCEDURE FOR spm1(:dept); DECLARE dept_proc_3 PROCEDURE FOR pref_user . spm2 (:dept); //</pre>
<p>w_sql</p> <p><b>Definition :</b></p> <ul style="list-style-type: none"> <li>validation_pb</li> <li>cb_1</li> <li>dw_1</li> <li>bonus_cursor</li> <li><b>emp_cur</b></li> <li>dept_proc</li> <li>dept_proc_2</li> <li>dept_proc_3</li> <li>dept</li> <li>i_member2</li> <li>s_lastmember</li> <li>s_lastmember2</li> <li>s_lastmember3</li> </ul>	<p>39</p> <p>40</p> <p>41</p> <p>42</p> <p>43</p> <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>	<pre>DECLARE dept_proc_3 PROCEDURE FOR pref_user . spm2 (:dept); // string s_member1 string sal_var int i_member2 ▶ DECLARE Emp_cur CURSOR FOR SELECT employee.emp_number, employee.emp_name FROM employee WHERE employee.emp_salary &gt; :Sal_var ; DECLARE Bonus_Cursor CURSOR FOR Select bonus_date from bonus;</pre>
<p>w_sql</p> <p><b>Definition :</b></p> <ul style="list-style-type: none"> <li>validation_pb</li> <li>cb_1</li> <li>dw_1</li> <li>bonus_cursor</li> <li>emp_cur</li> <li>dept_proc</li> <li>dept_proc_2</li> <li>dept_proc_3</li> <li>dept</li> <li>i_member2</li> <li>s_lastmember</li> <li>s_lastmember2</li> <li>s_lastmember3</li> </ul>	<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>	<pre>string s_member1 string sal_var int i_member2  DECLARE Emp_cur CURSOR FOR SELECT employee.emp_number, employee.emp_name FROM employee WHERE employee.emp_salary &gt; :Sal_var ; ▶ DECLARE Bonus_Cursor CURSOR FOR Select bonus_date from bonus;  string s_lastMember string s_lastMember2 string s_lastMember3</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><b>Definition :</b></p> <ul style="list-style-type: none"> <li>validation_pb</li> <li>cb_1</li> <li>dw_1</li> <li>bonus_cursor</li> <li>emp_cur</li> <li>dept_proc</li> </ul> <p><b>Stored procedure r</b></p> <ul style="list-style-type: none"> <li>spm1</li> <li>dept_proc_2</li> <li>dept_proc_3</li> <li>dept</li> <li>i_member2</li> </ul>	<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>
<p>bonus_cursor</p> <p>emp_cur</p> <p><b>DB items reference</b></p> <ul style="list-style-type: none"> <li>emp_name</li> <li>emp_number</li> <li>emp_salary</li> <li>employee</li> <li>dept_proc</li> <li>dept_proc_2</li> <li>dept_proc_3</li> <li>dept</li> <li>i_member2</li> <li>s_lastmember</li> <li>s_lastmember2</li> <li>s_lastmember3</li> </ul>	<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 &gt; :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

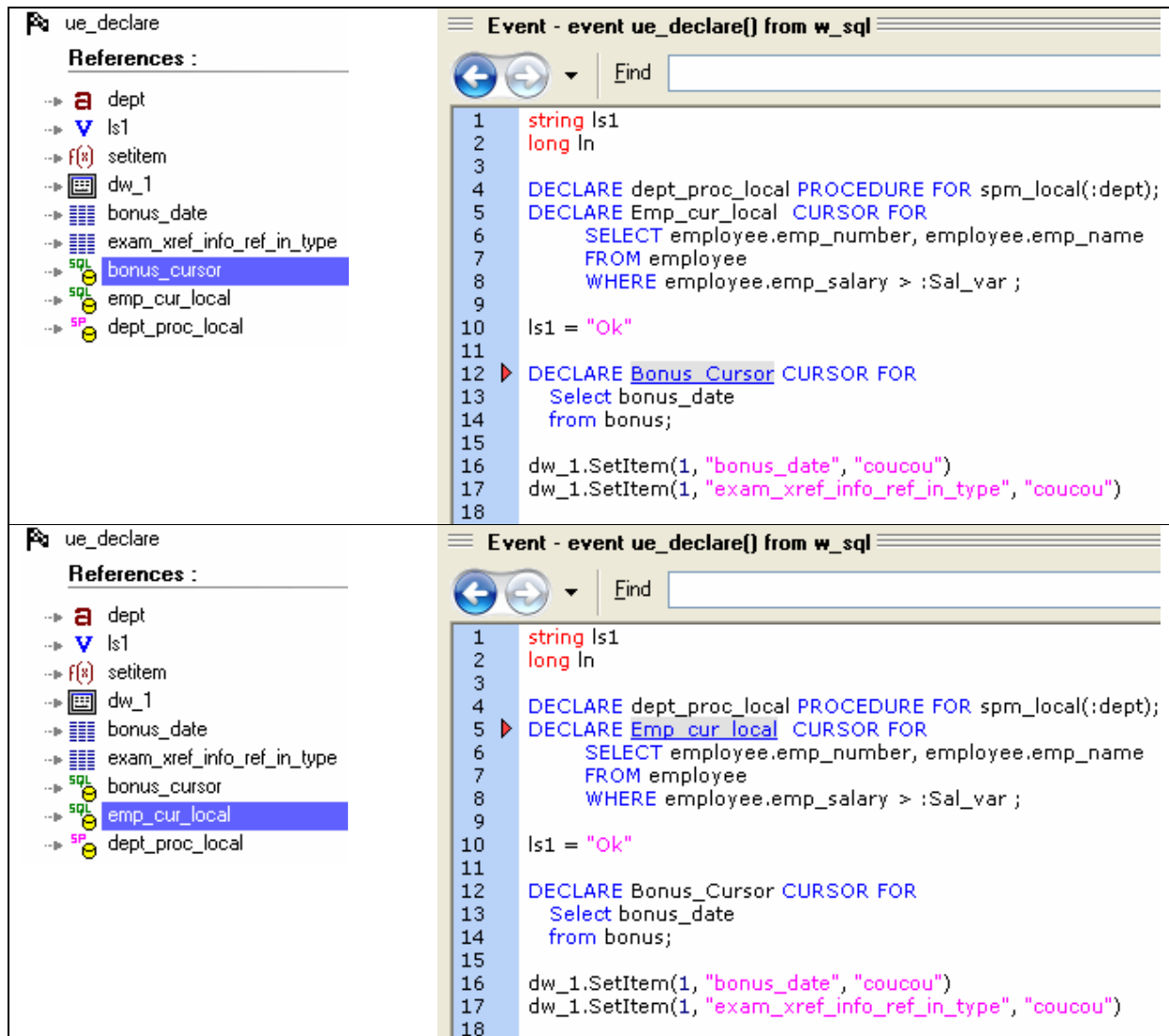
The macro “references” displays the logical cursors and procedures declared in PowerScript. For example:



**References :**

- dept string
- ls1 string
- setitem DATAWINDOWCONTROL
- dw\_1
- bonus\_date
- exam\_xref\_info\_ref\_in\_type
- bonus\_cursor
- emp\_cur\_local
- dept\_proc\_local

Again, the declaration is highlighted in the source code:



**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
```

**ue\_declare**

**References :**

- dept
- ls1
- setitem
- dw\_1
- bonus\_date
- exam\_xref\_info\_ref\_in\_type
- bonus\_cursor
- emp\_cur\_local
- dept\_proc\_local

**ue\_declare**

**References :**

- dept
- ls1
- setitem
- dw\_1
- bonus\_date
- exam\_xref\_info\_ref\_in\_type
- bonus\_cursor
- emp\_cur\_local
- dept\_proc\_local

**ue\_declare**

**References :**

- dept
- ls1
- setitem
- dw\_1
- bonus\_date
- exam\_xref\_info\_ref\_in\_type
- bonus\_cursor
- emp\_cur\_local
- 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 :**

- dept
- ls1
- setitem
- dw\_1
- bonus\_date
- exam\_xref\_info\_ref\_in\_type
- bonus\_cursor
- emp\_cur\_local

**DB items referenced :**

- emp\_name
- emp\_number
- emp\_salary
- employee

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 :**

- dept
- ls1
- setitem
- dw\_1
- bonus\_date
- exam\_xref\_info\_ref\_in\_type
- bonus\_cursor
- emp\_cur\_local
- 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(x) my_dll_func1	function string my_dll_func1(string, string) library "myDLL.dll" alias for external_func1
f(x) my_dll_func2	function string my_dll_func2(string, string) library "myDLL.dll" alias for external_func2
f(x) my_plsql_proc1	function string my_plsql_proc1(string) rpcfunc alias for sp_plsql1
f(x) my_plsql_proc2	function string my_plsql_proc2(string) rpcfunc alias for sp_plsql2
f(x) uf_method1	public function integer uf_method1(integer)
f(x) 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:

w_rpcfunc_test	20	
	21	type prototypes
	22	
Definition :	23	// déclarations de procédures stockées sous forme de méthode (RPCFUNC)
	24	//
validation_pb	25	▶ function string my_plsql_proc1 ( string toto ) rpcfunc alias for sp_plsql1
activate	26	function string my_plsql_proc2 ( string toto ) rpcfunc alias for sp_plsql2
clicked	27	
close	28	// déclarations de fonctions externes de DLL
open	29	//
f(x) my_dll_func1	30	function string my_dll_func1 ( string toto1, string toto2) library "myDLL.d
f(x) my_dll_func2	31	function string my_dll_func2 ( string toto1, string toto2) library "myDLL.d
f(x) my_plsql_proc1	32	
f(x) my_plsql_proc2		
f(x) uf_method1		
f(x) uf_method2		
w_rpcfunc_test	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
Definition :	27	
	28	// déclarations de fonctions externes de DLL
	29	//
validation_pb	30	▶ function string my_dll_func1 ( string toto1, string toto2) library "myDLL.d
activate	31	function string my_dll_func2 ( string toto1, string toto2) library "myDLL.d
clicked	32	
close	33	
open	34	
f(x) my_dll_func1	35	end prototypes
f(x) my_dll_func2	36	forward prototypes
f(x) my_plsql_proc1	37	public function integer uf_method1 (integer p1)
f(x) my_plsql_proc2		
f(x) uf_method1		
f(x) uf_method2		

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

<pre> open f(*) my_dll_func1 f(*) my_dll_func2 f(*) my_plsql_proc1 <b>References :</b> -&gt; sp_plsql1 f(*) my_plsql_proc2 f(*) uf_method1 f(*) uf_method2         </pre>	<pre> 20 21 type prototypes 22 23 // déclarations de procédures stockées sous forme de méthode 24 // (RPCFUNC) 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         </pre>
<pre> open f(*) my_dll_func1 <b>References :</b> -&gt; f(*) external_func1 f(*) my_dll_func2 f(*) my_plsql_proc1 f(*) my_plsql_proc2 f(*) uf_method1 f(*) uf_method2         </pre>	<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 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99         </pre>

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

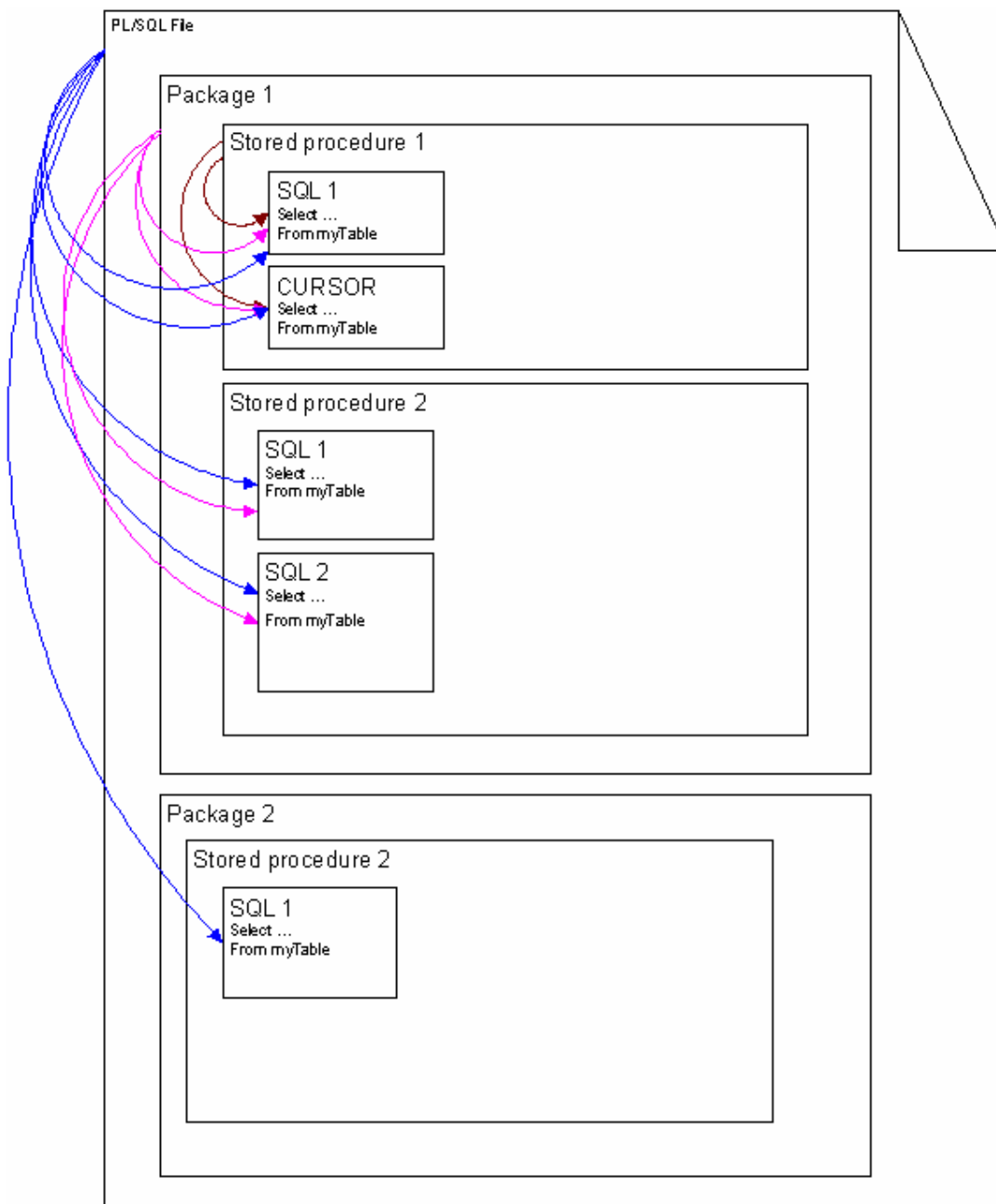
For DB code, Visual Expert does not display SQL items anymore. DB Objects are directly related to PL/SQL or T-SQL objects (packages, procedures, triggers, views, types, blocks...).

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:

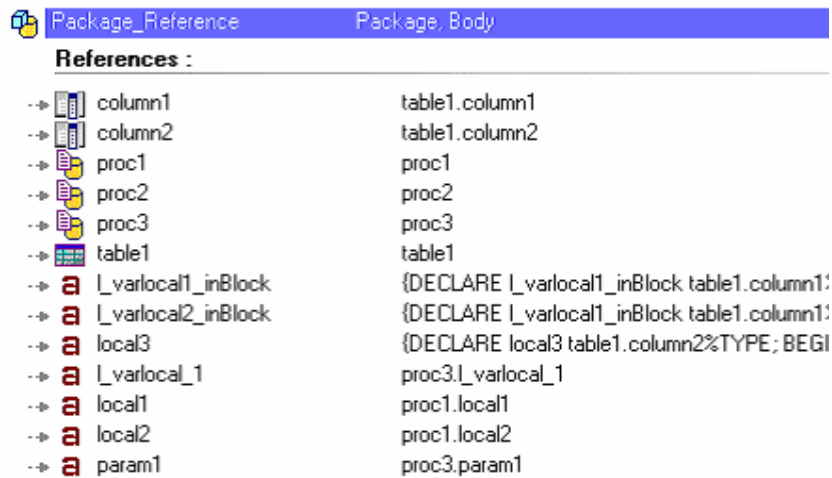


### 4.2.1. Oracle Package

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 'Package\_Reference' and 'Package, Body'. Below the header, a section titled 'References :' lists various database objects and their references. Each item is preceded by a small icon representing its type (e.g., a document icon for columns, a folder icon for procedures, a table icon for tables, and a red 'a' icon for local variables).

Item	Reference
column1	table1.column1
column2	table1.column2
proc1	proc1
proc2	proc2
proc3	proc3
table1	table1
_varlocal1_inBlock	{DECLARE _varlocal1_inBlock table1.column1:
_varlocal2_inBlock	{DECLARE _varlocal1_inBlock table1.column1:
local3	{DECLARE local3 table1.column2%TYPE; BEGI
_varlocal_1	proc3._varlocal_1
local1	proc1.local1
local2	proc1.local2
param1	proc3.param1

Package_Reference	44	-----
References :	45	PROCEDURE proc1
-> column1	46	AS
-> column2	47	
-> proc1	48	local1 table1.column1%TYPE;
-> proc2	49	local2 table1.column2%TYPE;
-> proc3	50	
-> table1	51	
-> l_varlocal1_inBlock	52	BEGIN
-> l_varlocal2_inBlock	53	
-> local3	54	SELECT column1, column2, proc1()
-> l_varlocal_1	55	FROM table1;
-> local1	56	
-> local2	57	proc2();
-> param1	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	
	90	END;
	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

<b>Package_Reference</b> <b>References :</b> -> column1 -> column2 -> proc1 -> <b>proc2</b> -> proc3 -> table1 -> _varlocal1_inBlock -> _varlocal2_inBlock -> local3 -> _varlocal_1 -> local1 -> local2 -> param1	42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62  74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94	IS  ----- PROCEDURE proc1 AS  local1 table1.column1%TYPE; local2 table1.column2%TYPE;  BEGIN  SELECT column1, column2, proc1() FROM table1;  proc2();  local1 := local2;  DECLARE   END;  ----- PROCEDURE proc2(param1 IN table1.column2 AS  local3 table1.column2%TYPE;  BEGIN  SELECT column1, column2 , proc1() FROM table1; proc2( proc3( proc1() ) );  END;  ----- PROCEDURE proc3(
---	--	--

Package_Reference	39	-----
<b>References :</b>	40	PACKAGE BODY Package_Reference
-> column1	41	IS
-> column2	42	-----
-> proc1	43	PROCEDURE proc1
-> proc2	44	AS
-> proc3	45	local1 table1.column1%TYPE;
-> table1	46	local2 table1.column2%TYPE;
-> a l_varlocal1_inBlock	47	BEGIN
-> a l_varlocal2_inBlock	48	SELECT column1, column2, proc1()
-> a local3	49	FROM table1;
-> a l_varlocal_1	50	proc2();
-> a local1	51	local1 := local2;
-> a local2	52	DECLARE
-> a param1	53	local3 table1.column2%TYPE;
	54	BEGIN
	55	local1 := local2;
	56	local1 := local1 + local2;
	57	local3 := local1 + local3;
	58	END;
	59	END;
	60	-----
	61	PROCEDURE proc3(
	62	param1 IN table1.column1%
	63	param2 IN table1.column2%
	64	)
	65	AS
	66	l_varlocal_1 table1.column1%type;
	67	BEGIN
	68	SELECT column1, proc1()
	69	FROM table1;
	70	BEGIN
	71	SELECT column1, column2, proc1()
	72	FROM table1;
	73	proc2( proc3( proc1() ) );
	74	END ;
	75	-----
	76	DECLARE
	77	l_varlocal1_inBlock table1.column1
	78	l_varlocal2_inBlock table1.column2
	79	BEGIN
	80	SELECT column1, column2, l_varloca
	81	FROM table1;
	82	proc2( proc3( proc1() ) );
	83	-----
	84	PROCEDURE proc3(
	85	param1 IN table1.column1%
	86	param2 IN table1.column2%
	87	)
	88	AS
	89	l_varlocal_1 table1.column1%type;
	90	BEGIN
	91	SELECT column1, proc1()
	92	FROM table1;
	93	BEGIN
	94	SELECT column1, column2, proc1()
	95	FROM table1;
	96	proc2( proc3( proc1() ) );
	97	END ;
	98	-----
	99	DECLARE
	100	l_varlocal1_inBlock table1.column1
	101	l_varlocal2_inBlock table1.column2
	102	BEGIN
	103	SELECT column1, column2, l_varloca
	104	FROM table1;
	105	proc2( proc3( proc1() ) );
	106	END ;
	107	-----
	108	PROCEDURE proc3(
	109	param1 IN table1.column1%
	110	param2 IN table1.column2%
	111	)
	112	AS
	113	l_varlocal_1 table1.column1%type;
	114	BEGIN
	115	SELECT column1, proc1()
	116	FROM table1;
	117	BEGIN
	118	SELECT column1, column2, proc1()
	119	FROM table1;
	120	proc2( proc3( proc1() ) );
	121	END ;
	122	-----
	123	DECLARE
	124	l_varlocal1_inBlock table1.column1
	125	l_varlocal2_inBlock table1.column2
	126	BEGIN
	127	SELECT column1, column2, l_varloca
	128	FROM table1;
	129	proc2( proc3( proc1() ) );
	130	END ;

 Package_Reference	44	-----
<b>References :</b>	45	PROCEDURE proc1
->  column1	46	AS
->  column2	47	
->  proc1	48	local1 table1.column1%TYPE;
->  proc2	49	local2 table1.column2%TYPE;
->  proc3	50	
->  table1	51	
->  _varlocal1_inBlock	52	BEGIN
->  _varlocal2_inBlock	53	
->  local3	54	SELECT column1, column2, proc1()
->  _varlocal_1	55	FROM table1;
->  local1	56	
->  local2	57	proc2();
->  param1	58	local1 := local2;
	59	DECLARE
	60	local3 table1.column2%TYPE;
	61	
	62	BEGIN
	63	local1 := local2;
	64	local1 := local1 + local2;
	65	local3 := local1 + local3;
	66	
	67	END;
	68	END;
	69	
	70	-----
	71	PROCEDURE proc2(param1 IN table1.column2
	72	
	73	AS
	74	
	75	
	76	
	77	
	78	
	79	
	80	

The screenshot shows a code editor with a 'References' pane on the left and a code view on the right. The 'References' pane lists various identifiers like column1, column2, proc1, proc2, proc3, table1, and several local variables. The code view shows a SQL procedure named proc3 with several nested SELECT and PROCEDURE calls. A red oval at the bottom highlights a list of line numbers: 48 54 86 95 104 109 118 123. These numbers correspond to references in the code that are not currently visible in the 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, a tree view under 'Content' shows a 'Package\_Reference' section with a 'Definition' for 'proc1' located at 'S:\VELOG\2008-11-07 15'. Below this, a 'References' list includes 'column1', 'column2', 'proc1', 'proc2', 'table1', 'local3', 'local1', and 'local2'. On the right, the SQL code for 'proc1' is shown, starting with 'PROCEDURE proc1 AS' and ending with 'END;'. The code includes local variable declarations for 'column1' and 'column2', a 'BEGIN' block with a 'SELECT' statement and a call to 'proc2()', and a 'DECLARE' section for 'local3'. A status bar at the bottom indicates '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:

<b>Content :</b> Package_Reference <b>Definition :</b> S:\VELOG\2008-11-07 proc1 <b>References :</b> 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>
<b>Content :</b> Package_Reference <b>Definition :</b> S:\VELOG\2008-11-07 proc1 proc2 proc3 <b>References :</b> column1 column2 proc1 proc2 proc3 table1 l_varlocal1_inBloc l_varlocal2_inBloc l_varlocal_1	93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115	<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. New source code view

### 5.1. Hyperlinks in the code

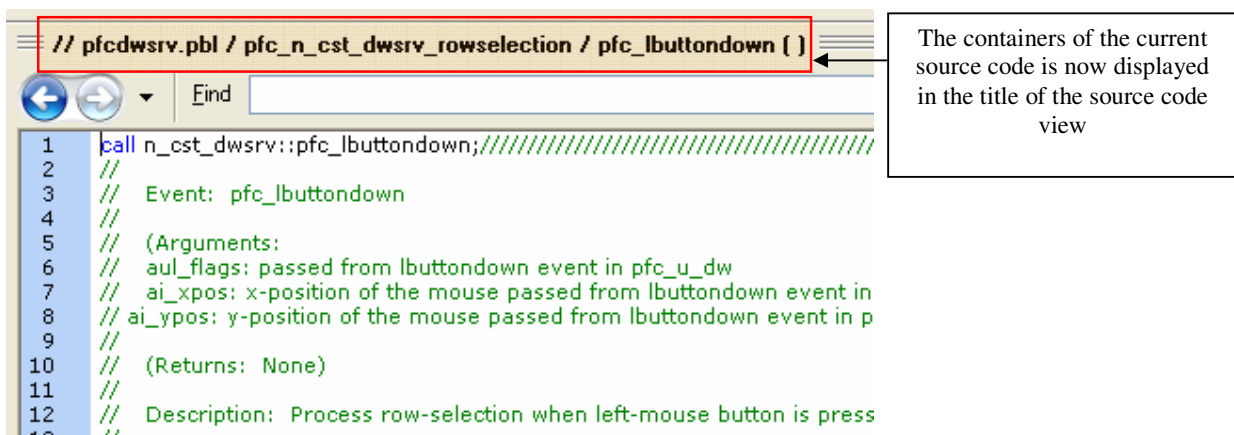
Hyperlinks are now available in the source code, both for methods and variables:

- A click on a referenced method will display its source code (Go to definition)
- A click on a referenced variable will display its declaration (Go to declaration)

### 5.2. Title of the source code view

The title of the source code view now displays the containers of the current code. After clicking on a hyperlink in the code, it helps to understand where is located the current code.

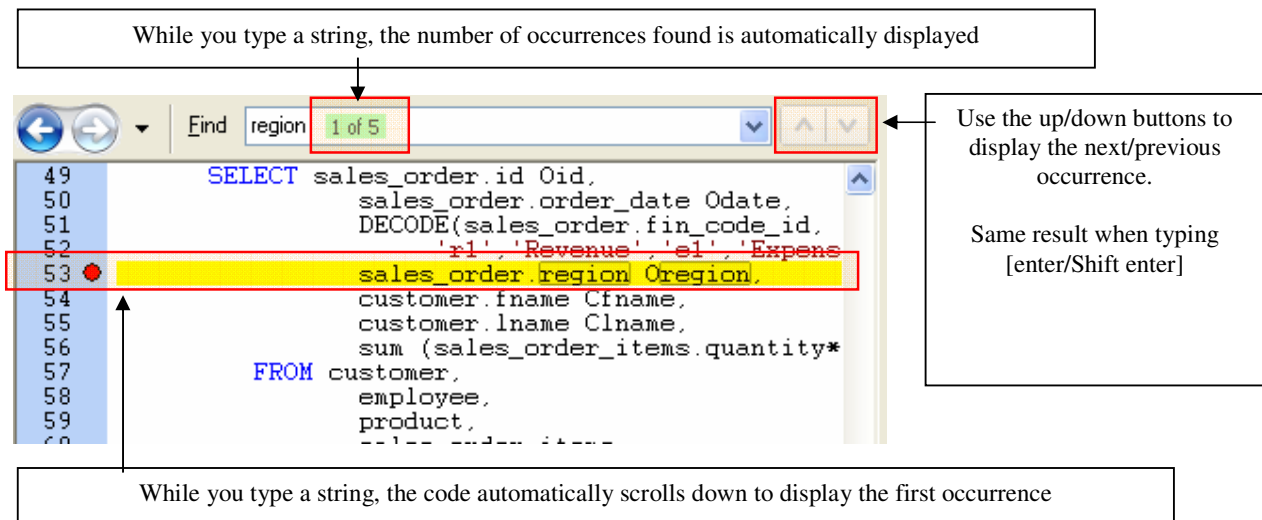
- For instance: `//{PBL}/{composant}/{contrôle}/{méthode}`
- Or `//{text file}/{Package}/{Procedure}`



### 5.3. Search in the source code view

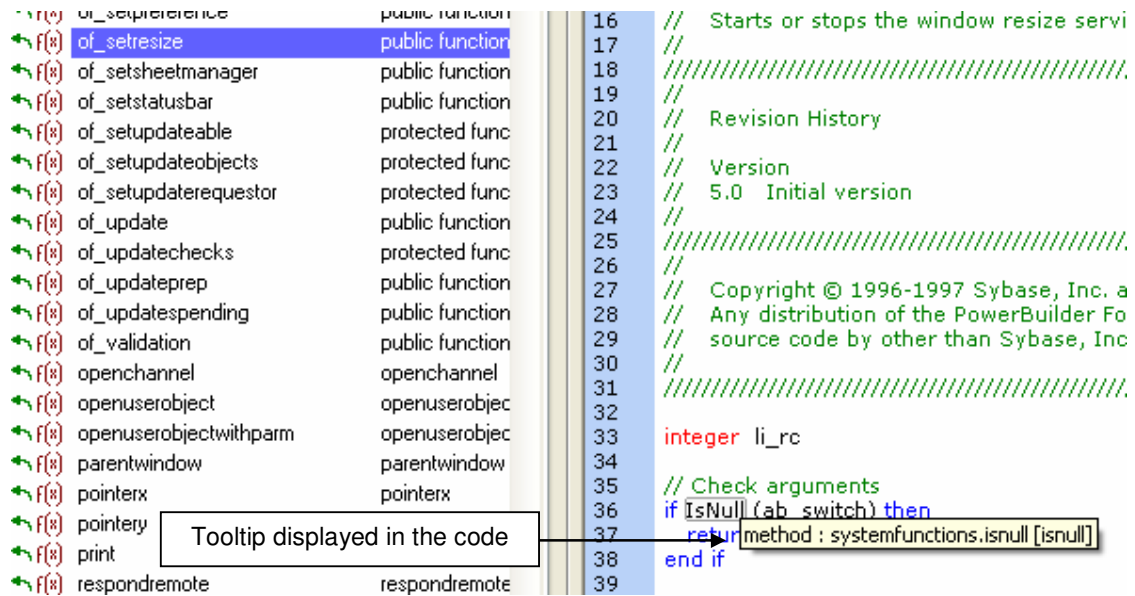
The search feature in the source code view has been redeveloped:

- When entering a string in the find field, Visual Expert indicates automatically the number of occurrences found in the current source code.
- The code automatically scrolls down to display the first occurrence found.
- Pressing [enter]/[Shift Enter] or clicking on the up/down button will go to the next/previous occurrence.



## 5.4. 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).



```
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
```

The image shows a code editor with a list of methods on the left and source code on the right. The method `of_setresize` is highlighted in blue. A tooltip box with the text "Tooltip displayed in the code" is positioned over the `return` keyword in the source code on line 37. The tooltip points to the `return` keyword in the code.

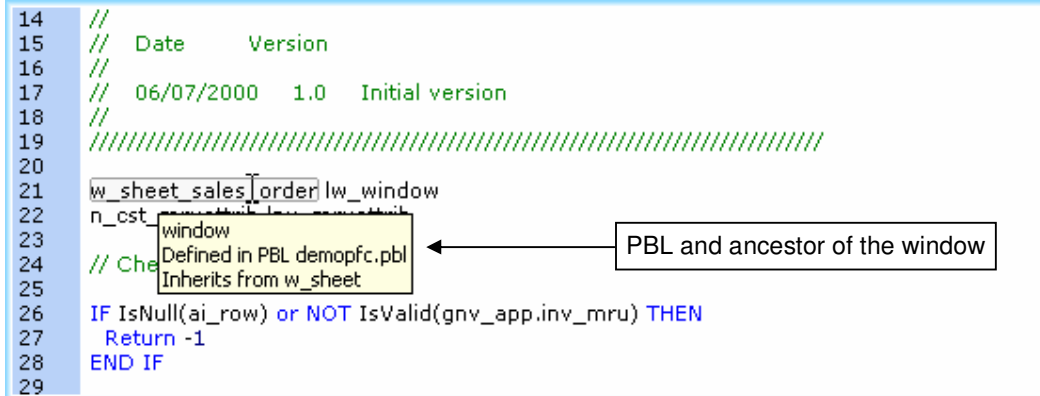
### 5.4.1. PowerBuilder objects

A tooltip will display detailed information about a PB Object:

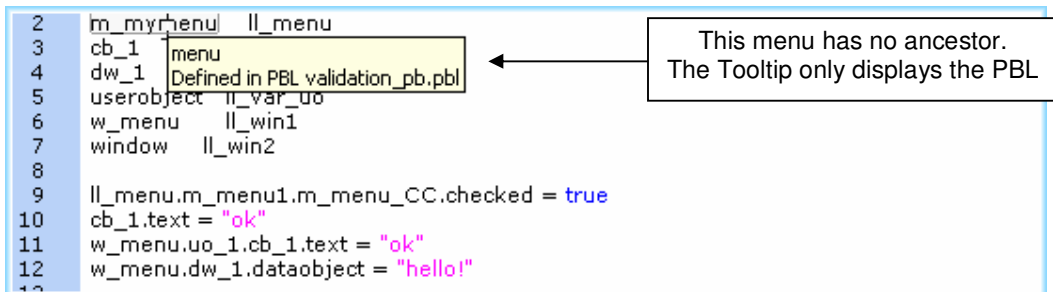
- Its type (window, userobject, menu ...)
- Its ancestor
- The PBL containing this object

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
24 // Che
25
26 IF IsNull(ai_row) or NOT IsValid(gnv_app.inv_mru) THEN
27     Return -1
28 END IF
29
```



```
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
```



### 5.4.2. 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 = "new"
13
14 uo_1.cb_1
15 dw_1.dataobject
16
17 m_MENU.text="ok"

```

userobject  
 Defined in PBL validation\_pb.pbl  
 Inherits from u\_test\_view1

Control's ancestor and PBL

```

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 = "new"
13
14 uo_1.cb_1
15 dw_1.dataobject
16
17 m_MENU.text="ok"

```

commandbutton  
 Control of w\_MENU.uo\_1  
 Inherits from u\_test\_view1.cb\_1

Same for nested controls

### 5.4.3. 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_mywindow ll_mywindow
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_mywindow.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.

datawindowcontrol  
dataobject='d\_dataobject1' defined in PBL validation\_pb.pbl

```
2 m_mywindow ll_mywindow
3 cb_1 ll_var_cb
4 dw_1 ll_var_dw
5 use
6 datawindowcontrol
7 dataobject='d_dataobject1' defined in PBL validation_pb.pbl
8 dataobject='d_proc1' defined in PBL validation_pb.pbl
9 ll_mywindow.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.

### 5.4.4. 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 local variable, type of m_myMENU
11 w_me Details for m_myMENU :
12 w_me menu
13
14 uo_1 Defined in PBL validation_pb.pbl
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 local variable, type of dw_1
6 w_menu ll Details for dw_1 :
7 window ll_w datawindowcontrol
8 dataobject='d_dataobject1' defined in PBL validation_pb.pbl
9 ll_menu.m_me dataobject='d_proc1' defined in PBL validation_pb.pbl
10 cb_1.text = "ok"
11 w_menu.uo_1.cb_1.text = "ok"
12 w_menu.dw_1.dataobject
13
14 uo_1.cb_1.text = "ok"

```

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'.

## 5.4.5. PowerBuilder methods

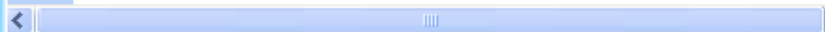
Functions tooltips will indicate :

- 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 message.of_setdoubleparm(public function integer of_getitem(long, ref n_cst_mruattrib)
34                             Belong to pfc_n_cst_mru
35
```

```
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
```



```
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     function_object
17     CASE "sybase", "mdic" Defined in PBL vg_security.pbl
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)
```

#### 5.4.6. 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             Column 'customer.cust_id'
146     SELECT Count(*) into Prod_num
147         FROM customer,
148             product,
149             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('-----');
```

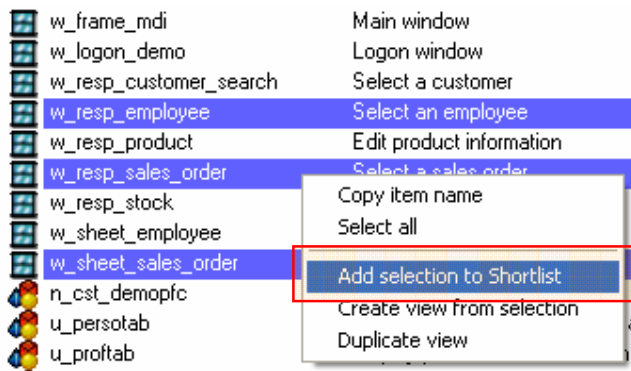
## 6. Miscellaneous

### 6.1. Technical requirements

- Polymorphism is now supported: it required Visual Expert exceptions to be declared
- 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.
- PB 11.5 is now supported (syntax analysis, integration with PB 11.5 IDE...)

### 6.2. Manage a Short-List of components

You can now select items in the treeview and in the source code and group them in a specific treeview tabpage called “ShortList”:



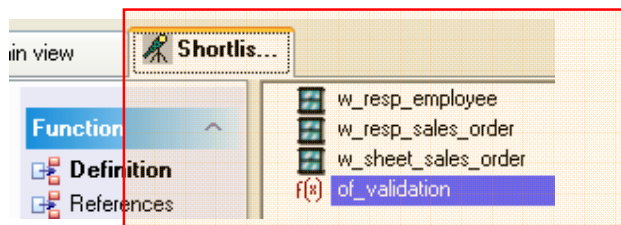
This menu option will add the selected objects in a new treeview tabpage called “Short List”

```
34
35 // Let Logical Unit of Work Service perform the functionality (
36 If IsNull(inv_luw) Or Not IsValid(inv_luw) Then of_SetLogica
37 If IsValid(inv_luw) Then
38   Return inv_luw.of_validation(app_control)
39 End If
40
41 Return -1
42
```

Context menu options:

- Add to shortlist**
- Locate
- Open 'of\_validation' in PB IDE

The function “of\_validation” will also be added in the “Short List”



When your selection is done, you can open the “Short List” and use the selected objects for documentation, impact analysis...