PL/SQL Developer plsqldoc Plug-In 1.2 User's Guide

1. Introduction

This document describes the plsqldoc Plug-In for PL/SQL Developer. This Plug-In allows you to generate HTML documentation for functions, procedures, packages, types, triggers, tables, views and materialized views. The documentation is derived from the definition and the comments of these objects, in a similar way as Javadoc for the Java programming language.

The Plug-In also allows you to quickly view object documentation from within the PL/SQL Developer IDE, by simply clicking on an object name in a SQL or PL/SQL source file, or from within the Object Browser. A global documentation index and the hyperlinks within each HTML document provide easy navigation.

The following example shows the documentation of a *LockRecord* and *EmpCount* function in a package:

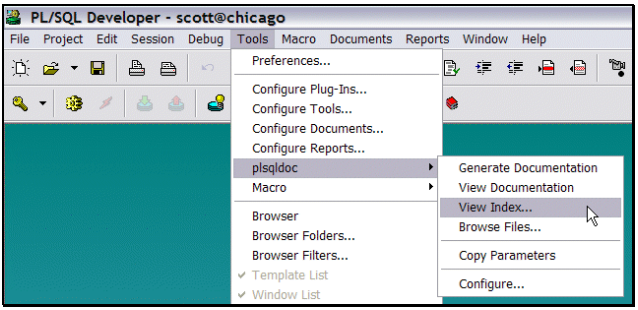
1.1 Benefits

The benefits of the plsqldoc Plug-In come from its generation capabilities, as well as the integration within the PL/SQL Developer IDE:

- **Increased documentation quality** – You no longer need to create and maintain separate documentation for your database objects, but merely need to write the appropriate comments within each object. The documentation generation is effortless, and this ensures that the documentation is always in sync with the latest developments. It also ensures a consistent layout of all documentation.
- **Increased productivity** – The user of the documentation can easily find a document and related documents, thanks to the tight integration with the PL/SQL Developer IDE and the hyperlinks. The author of the documentation benefits from the generation process.

1.2 Integration

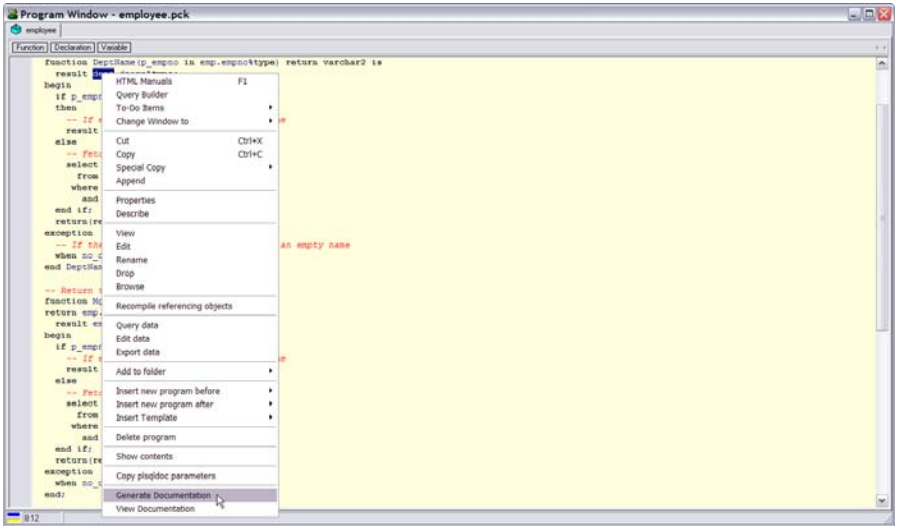
When the plsqldoc Plug-In is installed, you will have a new *plsqldoc* item in the *Tools* menu. This menu item contains all relevant functions:



The *Generate Documentation* and *View Documentation* items work on the object in the PL/SQL Editor of the current Program Window. You can right-click on the toolbar, select the *Customize* menu item, and add toolbar buttons for all frequently used plsqldoc functions:

This will make these functions easier accessible.

Right-clicking on an object name in a SQL or PL/SQL source or in the Object Browser will reveal 3 new functions to view and generate the documentation for the object under the mouse cursor, and to copy the parameters if a program unit in plsqldoc format:

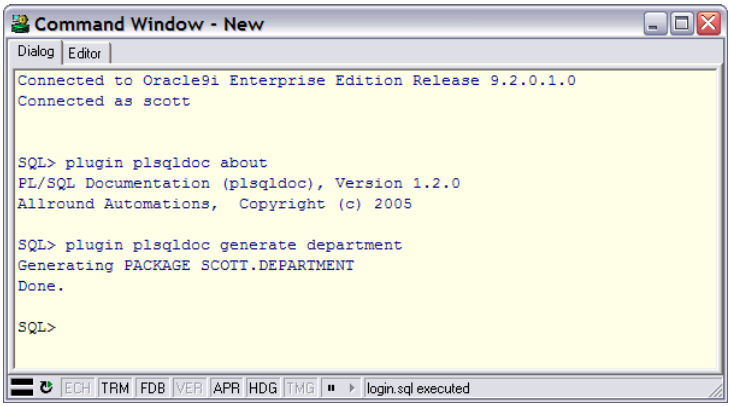


In the Command Window you will have the following new commands available when the plsqldoc Plug-In is installed:

Command	**Description**
plugin plsqldoc generate [body] <object1> <object2> ...	Generate the documentation for the specified object(s).
plugin plsqldoc view [body] <object> <object2> ...	View the documentation for the specified object(s).
plugin plsqldoc about	Show version information of the plsqldoc plug-in.
plugin plsqldoc configure	Show the plsqldoc configuration screen.
plugin plsqldoc index	Show the plsqldoc HTML index file.
plugin plsqldoc delete	Delete all HTML documentation files.
plugin plsqldoc rebuild	Rebuild the plsqldoc HTML index file.

These commands allow you to create a batch script that generates or refreshes your documentation. In case of a package or type, the default action is to generate the documentation from the specification. You can add the *body* keyword to indicate that the documentation should be generated from the package or type body. This keyword is ignored for all other object types.

The following example first shows the plsqldoc version information, and subsequently generates the documentation for the *Department* package:



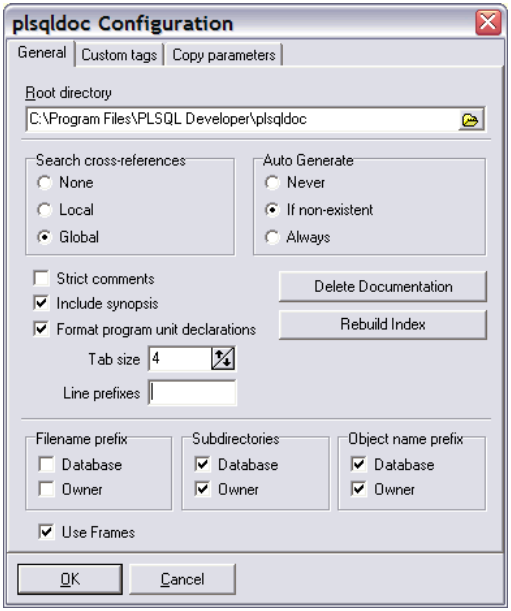
2. Installation

To install the plsqldoc Plug-In, simply run the supplied *setup.exe*. This will copy the Plug-In DLL (*plsqldoc.dll*) into PL/SQL Developer's *Plugins* directory. Make sure that you have write access to this directory when you run the setup program. After installation, you have the new functionality available as described in chapter 1.2.

System requirements
PL/SQL Developer 5.1.2 or later.

3. Configuration

Before you generate any documentation you should review the configuration options of the plsqldoc Plug-In. To do so, go to the *plsqldoc* submenu in the *Tools* menu, and select the *Configure* item. The following dialog will appear:



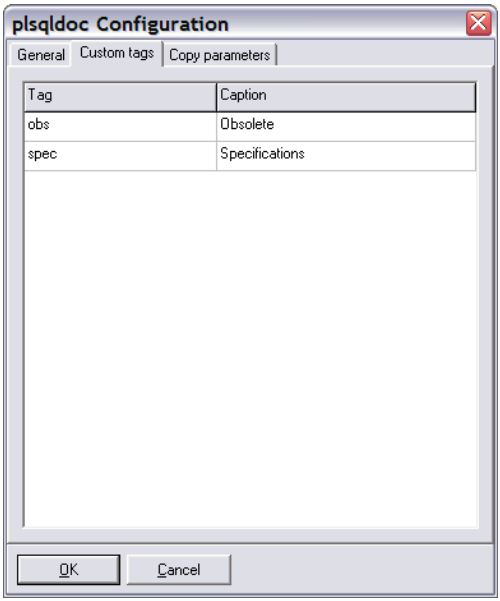
On the *General* page you see the following options:

- **Root directory** – This is where the HTML files of all documentation will be stored. The configuration will also be stored here, so that all documentation within a directory are generated with the same options. In a multi-user environment, you probably want to use a directory on a file server where all users have write access.
- **Search cross-references** – This option determines how plsqldoc will search for cross-references in a document. These cross-references will by translated to hyperlinks.
  - **None** – The document will not be searched for cross-references. All hyperlinks must be explicitly placed in the comments through *%link* and *%see* tags (see chapter 4.1).
  - **Local** – The document will be searched for cross-references within the same document. For example, a comment for a packaged procedure that references a type that is declared within the same package.
  - **Global** – All documents will be searched for cross-references. Note that the order in which you generate the documents will affect the included hyperlinks. If you first generate the documentation for a package that contains a table reference, but the document for this table is not yet generated, then this table reference will not result in a hyperlink.
- **Auto generate** – This option determines whether documentation is automatically generated when you right-click on an object to view it:

  - **Never** – The documentation is never automatically generated. An attempt to view documentation that has not yet been generated will lead to an error message.
  - **If non-existent** – The documentation is only generated if it does not yet exist.
  - **Always** – The documentation is always generated before viewing it. If you have a lot of complex objects, this option may be inhibited by performance. It does ensure that the documentation is always up-to-date though.
- **Strict comments** – When enabled, only comments that start with /** will be used as documentation. This allows you to explicitly separate documentation comments from other comments.
- **Include synopsis** – When enabled, the first sentence of a comment will be used as the synopsis for a documented item. The synopsis will be displayed in the global index of the documentation and in the index of packages and types.
- **Format program unit declarations** – When enabled, the declarations of program units in the documentation will be formatted by the PL/SQL Beautifier, according to the rules that you have defined.
- **Tab size** – Defines the number of spaces that will be inserted in the HTML file for a hard tab character in the documentation comments.
- **Line prefixes** – Defines a comma-separated list of character strings that will be treated as line prefixes. If, for example, all your comments start with || you can enter this information here. In the generated HTML these prefixes will be suppressed.
- **Filename prefix** – This option controls the prefix of the HTML files. You can include the database and object owner as prefix. If both are enabled, the filename will be *database.owner.object.html* (e.g. chicago.scott.employee.html).
- **Subdirectories** – This option controls whether subdirectories are created for databases and/or owners. If both are enabled, the directory will be *root\database\owner* (e.g. C:\Program Files\PLSQL Developer\plsqldoc\chicago\scott).
- **Object name prefix** – The option controls whether the database and owner are displayed as part of the object name in the documentation. This can be important context information when viewing a document. If both are enabled, the full object name will be *database.owner.object* (e.g. chicago.scott.employee).
- **Use Frames** – When enabled, the index and individual documents will be displayed in different frames on one HTML page. The index will be displayed on the left, with a list of all databases, users and objects. The documentation for the selected object will be displayed to the right. When disabled, the index will be displayed on a full page, and clicking on an object will replace the index.

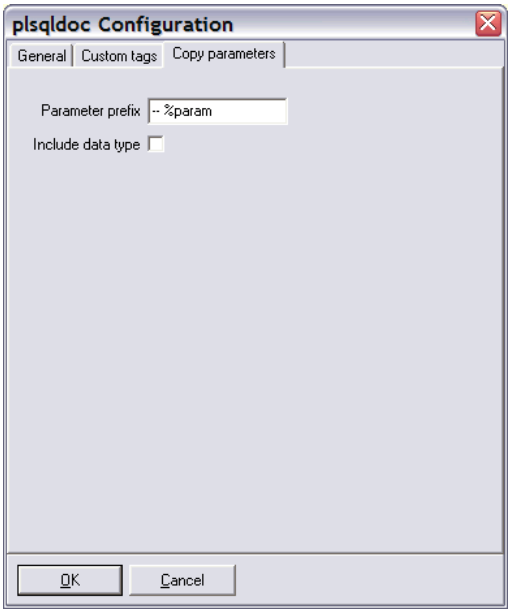
The dialog also provides a button to delete all document files, and a button to rebuild the index file from scratch.

On the *Custom tags* page you can define your own tags:



The example above allows you to use a *%obs* tag, which will translate to a documentation section with an *Obsolete* caption, and a *%spec* tag, which will translate to a documentation section with a *Specifications* caption. For more information about using tags, see chapter 4.1.

On the *Copy parameters* tab page you can define the behavior of the *Copy plsqldoc parameters* function:



- **Parameter prefix** – This prefix is inserted before each parameter that is copied from the function definition.
- **Include data type** – When enabled, the data type of each parameter is included. When disabled, only the name is included.

For more information about the *Copy plsqldoc parameters* function, see chapter 4.2.

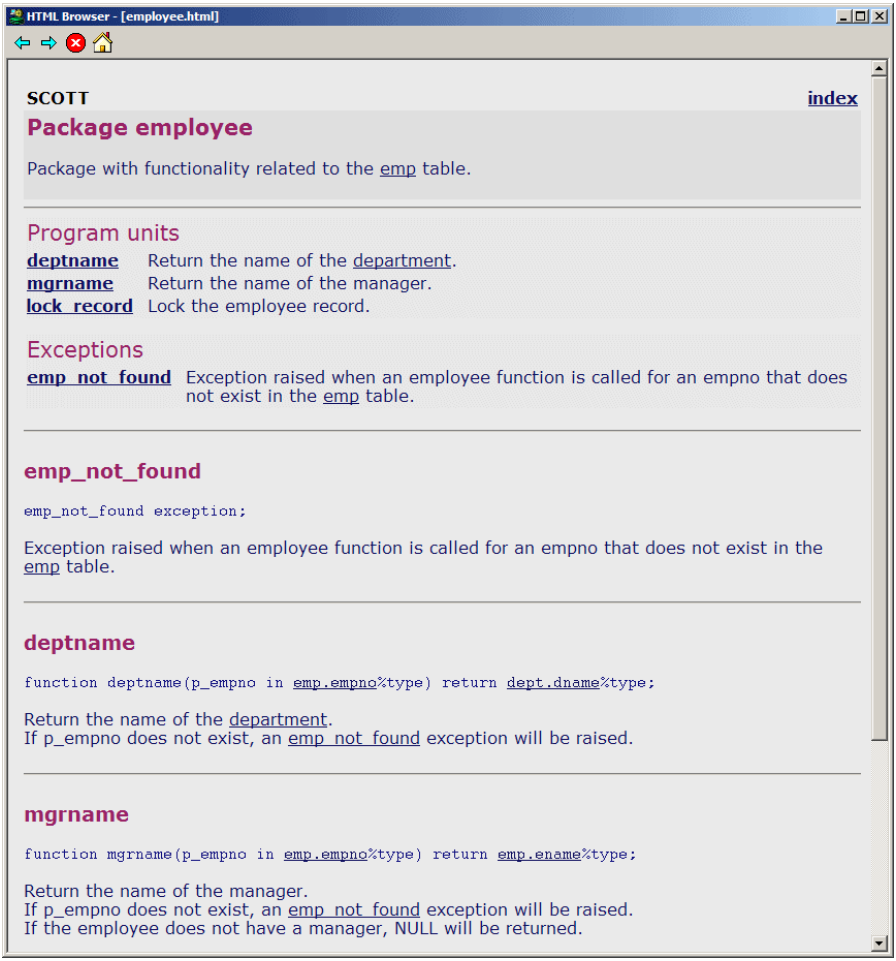
4. Writing documentation comments

Each declaration in the source of a PL/SQL program unit can be preceded by a comment section that will be included in the generated document. The comment section can contain tags that will be interpreted as specific documentation sections or hyperlinks. A comment section is either a single /* */ comment section or a section of subsequent -- comment lines finished by a non-comment line.

Even straightforward comments without any specific tags can lead to useful documents though. Let's assume a package specification like this:

```
create or replace package employee is

  -- Package with functionality related to the emp table.

  -- Exception raised when an employee function is called for an empno
  -- that does not exist in the emp table.
  emp_not_found exception;

  -- Return the name of the department.
  -- If p_empno does not exist, an emp_not_found exception will be raised.
  function deptname(p_empno in emp.empno%type) return dept.dname%type;

  -- Return the name of the manager.
  -- If p_empno does not exist, an emp_not_found exception will be raised.
  -- If the employee does not have a manager, NULL will be returned.
  function mgrname(p_empno in emp.empno%type) return emp.ename%type;

  -- Lock the employee record.
  -- If p_empno does not exist, an emp_not_found exception will be raised.
  -- The p_nowait indicates whether or not the procedure should wait for
  -- other locks to be released.
  procedure lock_record(p_empno  in emp.empno%type,
                        p_nowait in boolean);

end;
```

Plsqldoc will generate the following document for this package:



The first comment section in the package is interpreted as its description. The comment section preceding each package element is interpreted as the description of that element. The first sentence of each comment is interpreted as the synopsis.

Depending on the *Search cross-references* configuration option (see chapter 3), references to other objects will have been translated to hyperlinks.

As you can see, even without any plsqldoc-specific comments, you can still get a very useful document. To add more structure, you can include document tags as described in the next chapter.

4.1 Document tags

Document tags allow you to provide more structure for your documentation. For program units you can describe parameters, a return value, and exceptions that can be raised, for constants and variables you can describe the possible values, and so on.

Assume that we want to document the parameters of the following procedure:

```
procedure lock_record(p_empno  in emp.empno%type,
                      p_nowait in boolean);
```

We could precede this procedure with the following comment section:

```
-- Lock the emp record.
-- %param p_empno  The number of the emp record.
-- %param p_nowait Indicates whether or not the procedure should wait
--                 for other locks to be released.
procedure lock_record(p_empno  in emp.empno%type,
                      p_nowait in boolean);
```

The *%param* tags are used to describe each parameter. The tag is followed by the parameter name and the description. The following documentation would be generated for this procedure:

lock_record

```
procedure lock_record(p_empno in emp.empno%type, p_nowait in boolean);
```

Lock the emp record.

Parameters

p_empno The number of the emp record.
p_nowait Indicates whether or not the procedure should wait for other locks to be released.

Tags must be placed at the start of a comment line, except for the {%link} and {*} tags, which can appear anywhere in the comment section. The text of a tag finishes when a new tag is encountered, or at the end of the comment section. Leading spaces and returns will be ignored in the text. To include a return, use an HTML tag like <BR> or <P>. For /* */ comments sections, leading and trailing asterisks will be ignored on each line.

Other HTML tags that you can use are:

- <FONT> to specify the font.
- <B>, <I>, and <U> to use bold, italic and underlined text.
- <HR> to insert a horizontal line.
- <CODE> to insert example code using the courier font.
- <PRE> for preformatted text so that spaces and returns are preserved.
- <UL> and <LI> to include a bullet list.
- <TABLE>, <TD> and <TR> to include a table.

Including HTML tags in your code can improve the readability of the generated HTML documentation, but will decrease the readability of the actual PL/SQL code.

The following table describes all tags:

Tag	**Description**
%param <name> <text>	The description of a function or procedure parameter. Subsequent %param tags will be placed in one table.
%return <text>	A description of the return value of a function.
%value <name> <text>	A possible value for a package variable or object type attribute. Subsequent %value tags will be placed in one table.
%raises <name[.element]> <text>	A list of exceptions that can be raised by a program unit. Subsequent %raises tags will be placed in one table.
%see <name[.element[;n]]>	See also hyperlink. Subsequent %see tags will be placed in one table. To link to a specific overloading of a program unit, follow the name by a semi-colon and the overload index (1-based). For example: %see employee %see department.dname;2
%usage <text>	A text to describe the usage of the element.
%author <name>	The author of the object.
%version <text>	The version of the object.
{%link <name[.element[;n]]> [description]} or {%link <filename> [description]}	An explicit hyperlink. To link to a specific overloading of a program unit, follow the name by a semi-colon and the overload index (1-based). The optional description will be placed in the document. For example: The {%link department.name department name function} is... Will result in: The department name function is…
{*} <name> <text>	A bullet of a bullet list. Useful to describe possible values of a parameter or return value. For example: {*} 0 The function returned successfully {*} 1 The employee does not exist {*} 2 The employee record is locked
{%skip}	Skip this comment block for documentation.

The complete example in the next chapter demonstrates most of these tags.

In addition to these standard tags, you can also define your own custom tags on the *Custom tags* page of the configuration screen. See chapter 3 for more details.

Note: you can also use the @ or # character as tag-prefix instead of the % character, but this will cause problems if you want to compile your source files through SQL*Plus. In that case all your tags will be treated as start commands, and you will get a lot of error messages (e.g. unable to open file "param.sql").

4.2 Generating parameter tags

To help you create a list of tags for a parameter list, you can select the list in the editor (including the parentheses), right-click on it, and select *Copy plsqldoc parameters* from the popup menu. Now the tags for this parameter list are placed on the clipboard, which you can subsequently paste in the corresponding comment section for documentation. This way you merely need to add a description to each parameter.

The exact format of the parameter tags can be configured as described in chapter 3.

4.3 A complete example

```
create or replace package Department is

  -- Package with functionality related to the Dept table.
  -- Most functions require a p_DeptNo parameter, which
  -- is the value of the DeptNo column in the {%link Dept}
  -- table.

  -- Value for p_Wait parameters, indicating to wait
  -- until a lock is released.
  c_Wait constant boolean := true;
  -- Value for p_Wait parameters, indicating not to
  -- wait until a lock is released.
  c_NoWait constant boolean := false;

  -- Exception raised when an invalid order is specified
  e_InvalidOrder exception;
  -- Exception raised when a record is locked
  e_RecordLocked exception;

  -- A department record. It contains all columns of the
  -- {%link Dept} table.
  subtype t_DeptRecord is Dept%rowtype;

  -- A cursor type for parameters that return a set of
  -- departments.
  type t_DeptCursor is ref cursor return Dept%rowtype;

  -- Open a cursor to select departments in a particular order.
  -- %param p_cursor The cursor that will be opened for all
  --                 Dept records in the given order
  -- %param p_order  The order of the Dept records. Valid values are:
  --                 {*} 'DEPTNO' Records are ordered by DeptNo.
  --                 {*} 'NAME'   Records are ordered by name.
  --                 {*} 'LOC'    Records are ordered by loc. The secondary
  --                              sort column is DeptNo.
  --                 {*} NULL     The result set  will not be ordered
  --                              (default).
  -- %raises e_InvalidOrder Raised when the value of p_order is invalid.
  procedure SelectRecords(p_Cursor in out t_DeptCursor,
                          p_Order  varchar2 default null);

  -- Lock a Dept record.
  -- %param p_DeptNo The number of the department record to lock.
  -- %param p_Wait   Wait for an existing lock to be released?
  --                 {*} c_Wait   Wait until the record is available.
  --                 {*} c_NoWait Don't wait, and raise e_RecordLocked
  --                              if the record is locked.
  -- %raises e_RecordLocked Raised when the record is locked and
  --                        p_Wait = c_Wait.
  procedure LockRecord(p_DeptNo Dept.DeptNo%type,
                       p_Wait   boolean default c_Wait);

  -- Return the number of employees in the {%link Emp} table that
  -- belong to this department.
  -- %param  p_DeptNo The number of the department.
  -- %return The number of employees
  function EmpCount(p_DeptNo Dept.DeptNo%type) return integer;

  -- Return the employee number of the manager of the department.
  -- %param  p_DeptNo The number of the department.
  -- %return The number of the employee (Emp.EmpNo) that manages
  --         this department (Emp.job = 'MANAGER'). If the
  --         department does not have a manager, NULL is returned.
  function Manager(p_DeptNo Dept.DeptNo%type) return Emp.EmpNo%type;

end Department;
```

(continued on next page…)

(continued on next page…)


HTML Browser - [Department.html]
LockRecord
procedure LockRecord(p_DeptNo Dept.DeptNo%type,
p_Wait boolean default c_Wait);
Lock a Dept record.
Parameters
p_DeptNo The number of the department record to lock.
p_Wait Wait for an existing lock to be released?
c_Wait Wait until the record is available.
c_NoWait Don't wait, and raise e_RecordLocked if the record is locked.
Raises
e_RecordLocked Raised when the record is locked and p_Wait = c_Wait.
EmpCount
function EmpCount(p_DeptNo Dept.DeptNo%type) return integer;
Return the number of employees in the Emp table that belong to this department.
Parameters
p_DeptNo The number of the department.
Returns
The number of employees
Manager
function Manager(p_DeptNo Dept.DeptNo%type) return Emp.EmpNo%type;
Return the employee number of the manager of the Department.
Parameters
p_DeptNo The number of the department.
Returns
The number of the employee (Emp.EmpNo) that manages this department (Emp.job = 'MANAGER'). If the department does not have a manager, NULL is returned.

5. Customizing the documentation appearance

If you want to change the appearance of the various fonts and tables of the generated documents, you can edit the *plsqdoc.css* file in the plsqldoc root directory (see chapter 3). This is the CSS stylesheet that is used in the generated HTML files. Changes will immediately be reflected in the documentation without the need to regenerate anything. The stylesheet file contains a brief description of each style that it defines.

If you are not familiar with Cascading Style Sheets, please visit http://www.w3.org/TR/REC-CSS2 for more information.

You can additionally change the appearance of the index file. In the *index.html* in the plsqldoc root directory you can create your own header and footer, above the *<!--plsqldoc_start-->* comment tag, and below the *<!--plsqldoc_end-->* tag. Here you can add your own company logo, insert other relevant hyperlinks, or add an additional description of the documentation.