Oracle Forms Look & Feel Project

Developer guide

Francois Degrelle
http://fdegrelle.over-blog.com/
forms.pjc.bean@free.fr

Oracle Developer Forms Runtime - Web

Fenêtre

Forms Look & Feel demo

Block | Dynamic | Items | HTML Map | Charts | Image Spinner

Table-block title

Empno	Ename	Job	Mgr	Hiredate	Sal	Comm
7369	SMITH	CLERK	7902	17-DEC-1980	915	10
7499	ALLEN	SALESMAN	7698	20-FEB-1981	1600	300
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500
7566	JONES	MANAGER	7839	02-APR-1981	2975	
7654	MARTIN	SALESMAN	7698	28-SEP-1981	1250	1400
7698	BLAKE	MANAGER	7839	01-MAY-1981	2850	
7782	CLARK	MANAGER	7839	09-JUN-1981	2450	
7788	SCOTT	ANALYST	7566	09-DEC-1982	3000	
7839	KING	PRESIDENT		17-DEC-1980	6000	
7844	TURNER	SALESMAN	7698	08-SEP-1981	1500	0

Theme #1 | Theme #2 | Theme #3

Record: 3/14 | ...

Oracle Developer Forms Runtime - Web

Fenêtre

Forms Look & Feel demo

Block | Dynamic | Items | HTML Map | Charts | Image Spinner

Graphics

Frames

Menus

Popups menus

Popups messages

Dynamic Items

Text ar...

For...

Swi... value

Checkbox

0 25 50 75 100

Get image...

Obj:slider Name:slider Type:mouseevent val:exited

Record: 1/1 | ...

Content

Introduction

When you are using a product to develop a piece of software, you often (always?) have to push it to its limits. Sometimes, these limits are not enough, and you are stuck, then it requires a lot of imagination to find workarounds.

Even though your application is just fine, answering the end-user questions perfectly and providing the right functions, there is often (always?) a weak point, and sometimes, the weak point is the “design”, the “look”.

As is well known, you can not please everyone, so that you will probably have to manage different feelings ranging from “Whow, beautiful!” to “Arg, it is so ugly!”

Mixing all these points is a good way for introducing the Forms Look and Feel Project.

What better way to please everyone by letting them select their own look ?

Is it the “raison d'être” of the Cascading Style Sheets, so that, what about using the CSS system in an Oracle Forms application?

This is exactly what the Forms Look and Feel Project does. Using a CSS file to adapt the look of a Forms Application at runtime.

Warning

This tool does not come from Oracle and is not supported by the Oracle Support.

Do not open Service Request on Metalink to ask questions about it.

Instead, send emails to the following email address:

forms.pjc.bean@free.fr

Or ask questions on the dedicated forum:

Discussion Forum

Use also the email or the forum to report any problem you could encounter during installation, configuration, using or simply understanding the tool.

There is no license attached to this project. It is open source, and you can use, modify and distribute it as you want without any authorisation of any kind.

LAF Project presentation

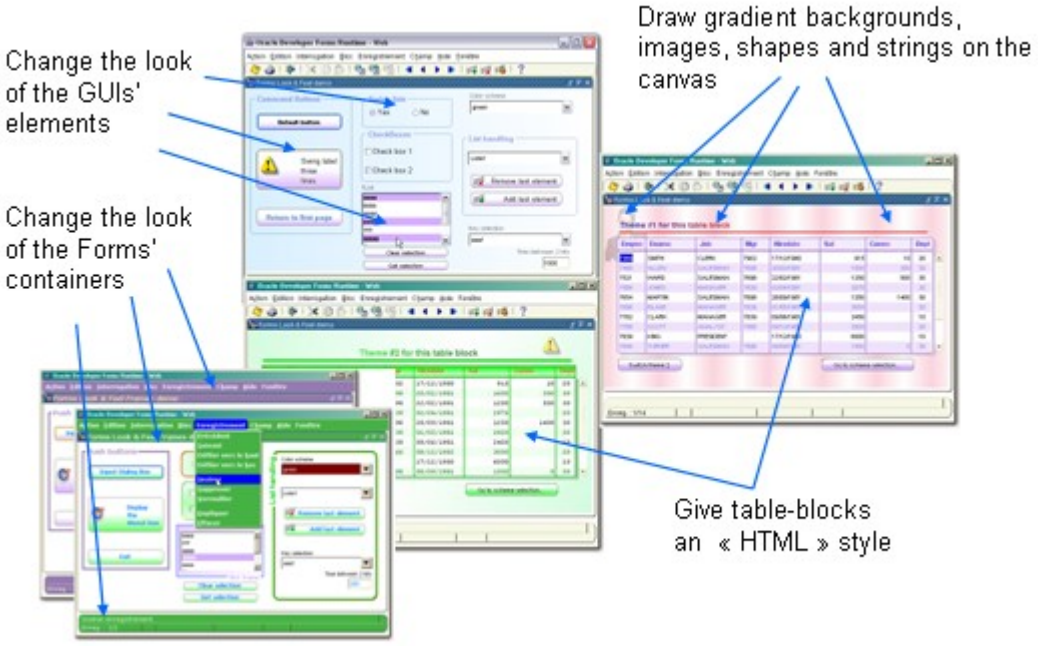
The Look and Feel Project is a set of tools that permits decorating a Forms module at runtime.

All the decoration information is stored and read from an external CSS file, then applied to the Forms elements at runtime.

It has bean created to answer the following questions:

- Change easily the look of the Forms application.
- Have a more «HTML» look.
- Externalize the graphical information.
- Add some missing functions

Easy change of the look at runtime



A lot of areas/widgets can be decorated by the tool:

- Window caption
- Menu bar
- Canvas
- Tab canvas

- Status bar
- Alert box
- Items

Gives the table-blocks a more HTML look

Empno	Ename	Job	Mgr	Hiredate	Sal	Comm	Dept
7369	SMITH	CLERK	7902	17-DEC-1980	915	10	20
7499	ALLEN	SALESMAN	7698	20-FEB-1981	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30
7566	JONES	MANAGER	7839	02-APR-1981	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-1981	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-1981	2850		30
7782	CLARK	MANAGER	7839	09-JUN-1981	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-1982	3000		20
7839	KING	PRESIDENT		17-DEC-1980	6000		10
7844	TURNER	SALESMAN	7698	08-SEP-1981	1500	0	30

Add a few new functions:

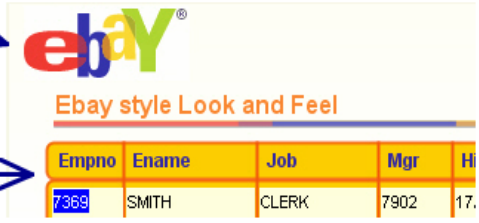
- Handle menus at runtime (add, enable, disable, show, hide, remove options).
- Handle frames at runtime (add, move, modify, hide).
- Play pre-loaded sounds.
- Receive external asynchronous messages.
- Draw texts anywhere on the screen.
- Display Swing JTable LOVs, Alerts and input dialogue boxes.
- JColorChooser.

The tool is made up of a PL/SQL library (laf.pll), a set of Java Beans and PJCs grouped in a jar file (laf_xxx.jar), and an external file containing the information tags (CSS).

- The PL/SQL library contains functions and procedures needed to decorate the canvases and the block tables.
- The jar file contains the beans needed to paint over the canvas and overload the standards forms widgets (buttons, check-boxes, radio groups and lists).
- The CSS file contains the tags used to describe how each Forms element will be decorated.

Because the graphical information is read from a given CSS file, it is easy to change the look and feel of the application without modifying the form modules. With it, you can really externalize the look of the Forms application by separating the functional implementation to the graphical presentation.

```
canvasEbay {
     type:canvas
     gradient-colors:r255g255b204,r255g255b255
     gradient-vcycle:/2
     gradient-hcycle:0
     image1:/ebay.jpg,10,1,.8
     image2:/ebay_line.jpg,30,74,.6,560,5
   }
tableHeaderEbay {
    type:header
    font-family: Arial;
    font-size:12
    font-weight:bold
    frame-color:r250g83b12
    inside-color:r255g204b0
    font-color:r0g0b204
    shade-color:r255g255b204
    frame-width:2
    frame-rounded-border: 10
    transparency:.8
    text-align:left
    text-align-offset:5
    }
```

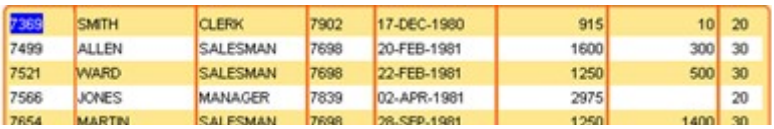


Another goal is to give the table blocks a more "HTML" look. Each block is divided in three sections, that can be decorated separately:

- the title section
  it can contains a text and a line to underline it

- the header section
  Contains many tags to decorate the head section of the table.

- the body section
  contains many tags to decorate the table body.



Go to the CSS file section to see a complete description of all the tags you can use.

System configuration

- Copy of the JAR file

The zip file of the project contains a **/Java/JAR** sub-folder with four files:

- **laf_902.jar** if you use the 9.0.2 or 9.0.4 version
- **laf_1012.jar** if you use the 10.1.2 version
- **laf_10123.jar** if you use the 10.1.2.3 version
- **laf_11112.jar** if you use the 11g version

Depending of the Forms version you use, copy the corresponding JAR file to your **/forms/java** folder.

> Note:
> the provided JAR files are already signed.
> If you rebuild them, you will have to sign them.

For Forms 11g the /java directory is located in the <middleware_home>/forms/java directory.

- Update of the **formsweb.cfg** configuration file

You also have to add the JAR file name to the **archive** tag of your **/forms/server/formsweb.cfg** configuration file.

```
...
archive=frmall.jar,laf_1012.jar
...
```

> note:
> We update the archive tag and not the archive_jini because this tool uses methods available in the 1.4 JRE, so it won't run with the Jinitiator.

> Note:
> If you intend to play MP3 sound files, add also the **jl1.0.jar** file in your /forms/java folder and to the **archive** tag.
> It can be loaded from the following URL:
> http://prdownloads.sourceforge.net/javalayer/jlayer1.0.zip?download

For Forms 11g the formsweb's directory is located in the <middleware_home>\user_projects\domains\ClassicDomain\config\fmwconfig\servers\WLS_FORMS\applications\formsapp_11.1.1\config\ directory

Understanding the components

The CSS file

To achieve the goal of separating the decoration information from the Forms module, each element's decoration attributes are stored in an external file, allowing the developer to change the Look and Feel at any time without any module recompilation.

It looks like a “real” CSS file, even if the tags are Forms dedicated.

It contains sections of five different types:

- **type:canvas** dedicated to tags used to decorate a canvas
- **type:title** dedicated to tags used to decorate the table-block title
- **type:header** dedicate to tags used to decorate the table-block header
- **type:body** dedicate to tags used to decorate the table-block body
- **type:gui** dedicated to tags used to decorate the other Forms areas

The tags included in sections of type **canvas** are read by the *Paint_Canvas()* procedure stored in the **laf.pll** PL/SQL library.

The tags included in sections of type **title, header** and **body** are read by the *Paint_Block()* procedure stored in the **laf.pll** PL/SQL library.

The tags included in sections of type **gui** are read by the *Set_GUI_Properties()* procedure stored in the **laf.pll** PL/SQL library.

Doc: The description of all available tags can be read from the CSS file properties documentation.

The laf.pll PL/SQL library

It contains the functions and the procedures needed to open the CSS file, read the tags from it, then perform the drawing job through the associated Java Bean's methods.
Here are the four main procedures the developer will use in the Forms triggers:

- Open and read the CSS file.

  ```
  PKG_LOOK_AND_FEEL.Open_Css()
  ```

- Draw objects on the canvas.

  ```
  PKG_LOOK_AND_FEEL.Paint_Canvas()
  ```

- Decorate the given table-block.

  ```
  PKG_LOOK_AND_FEEL.Paint_Block()
  ```

- Set global properties for every GUI widgets.

  ```
  PKG_LOOK_AND_FEEL.Set_GUI_Properties()
  ```

Description of the functions and procedures stored in the PKG_LOOK_AND_FEEL package:

- Open the CSS file

```
Function   Open_CSS( PC$Filename IN Varchar2 )  -- CSS filename
Return Boolean ;
```

This function is used to open the given CSS file.
If the file cannot be opened, it returns FALSE, and the other painting methods won't be accessible.
You must provide the full pathname.
A Forms parameter – *PM$CSS_FILENAME* - is provided in the Forms template (LAF_TEMPLATE.fmb) and also in the laf.olb Object Library. It can be used to set the file name through the *Call_Form()*, *Open_Form()* and *New_Form()* built-ins.

e.g.

```
If PKG_Look_And_Feel.Open_Css(:PARAMETER.PM$CSS_FILENAME) Then
   -- ok, we can use the painting methods --
   ...
End if;
```

- Paint the canvas

```
Procedure  Paint_Canevas
(
  PC$Class       IN Varchar2,     -- Canevas CSS class name
  PC$BeanName    IN Varchar2      -- associated bean area
) ;
```

This procedure is used to paint the given canvas where the *PC$BeanName* Bean Area is located. Every tag contained in the *PC$Class* CSS section name is applied on the canvas. The CSS class name given must be of type: **canvas**

e.g.

```
  -- paint the canevas that supports the :CTRL.BEAN Bean Area --
  -- with the content of the .maincanvasOracle CSS section
  PKG_LOOK_AND_FEEL.Paint_Canevas('.maincanvasOracle', 'CTRL.BEAN' ) ;
```

- Paint the table-block

```
Procedure  Paint_Block
(
  PC$Block       IN Varchar2,                -- the block name to decorate
  PC$BeanName    IN Varchar2,                -- the associated bean area
  PC$VA_Name     IN Varchar2,                -- the visual attribute associated
  PC$HeadClass   IN Varchar2,                -- the table header CSS class name
  PC$BodyClass   IN Varchar2,                -- the table body CSS class name
  PC$TitleClass  IN Varchar2 Default Null,   -- the table title CSS class name
  PC$Title       IN Varchar2 Default Null,   -- the block title
  PB$ScrollBar   IN Boolean  Default True,   -- scrollbar exists on block
  PB$SortBlock   IN Boolean  Default Null    -- can sort the table_block
) ;
```

This procedure is used to paint the given block.

The *PC$VA_Name* parameter (*) is the name of the Visual Attribute used to colour each other row.
The *PC$HeadClass* parameter indicates which CSS section to use to paint the table header. It must be of type: **header**
The *PC$BodyClass* parameter indicates which CSS section to use to paint the table body. It must be of type: **body**
The *PC$TitleClass* parameter indicates which CSS section to use to paint the table title. It must be of type: **title** and is not required.
If indicated, the *PC$Title* contains the title text you want to draw.
The *PB$ScrollBar* parameter indicates if you allow the procedure to move the scrollbar. Sometimes, the procedure may have to move the items between each other to draw the lines, then the addition of these moves can need to push the scrollbar away. If this parameter is set to FALSE, the scrollbar won't be moved.
The *PB$SortBlock* parameter indicates if the user can sort the block by clicking the table header.

(*) Since the 1.3.8 version, PC$VA_Name argument is obsolete, as you can define it in the CSS file.

e.g.

```
PKG_LOOK_AND_FEEL.Paint_Block
(
  PC$Block      => 'EMP'
 ,PC$BeanName   => 'CTRL.BEANTAB'
 ,PC$VA_Name    => :PARAMETER.PM$VA
 ,PC$HeadClass  => '.tableHeaderOracle'
 ,PC$BodyClass  => '.tableBodyOracle'
 ,PC$TitleClass => '.tableTitleOracle'
 ,PC$Title      => 'Oracle BLAF Look and Feel'
 ,PB$ScrollBar  => True
) ;
```

- Paint the other areas

```
Procedure  Set_GUI_Properties
(
  PC$Class       IN Varchar2,                -- GUIs CSS class name
  PC$BeanName    IN Varchar2                 -- the associated bean area
) ;
```

This procedure is used to decorate the other Forms elements, like Window caption, menu bar, status bar, elements and so on.
It need the CSS section name and the Bean Area name.
The CSS class name must be of type: **gui**

e.g.

```
-- set the global GUI properties --
PKG_LOOK_AND_FEEL.Set_GUI_Properties( '.GUIPropertiesOracle',
                                      'CTRL.BEAN' ) ;
```

- Write the tags in the CSS file

```
Function   Write_CSS ( PC$Filename IN Varchar2 ) -- CSS filename
Return Boolean ;
```

This function is used to write the tags stored in memory to a file.
It is used by the **css_updater.fmb** sample dialog created to update the tags in a wysiwig way.

- Read a specific tag value

```
Function   Get_Tag_Value
(
  PC$Section  IN Varchar2,                 -- CSS section name
  PC$TagName  IN Varchar2,                 -- CSS tag name
  PC$Default  IN Varchar2 Default 'none'   -- default value
) Return Varchar2 ;
```

It is used to return the value of the given section/tag name. If the tag is not found, it returns the PC$Default value if given, else it returns NULL.

e.g.

```
-- get the value of the font-family tag in the .tableHeaderOracle section --
LC$Value := Get_Tag_Value( '.tableHeaderOracle', 'font-family');
```

- Write a specific tag value

```
Procedure Set_Tag_Value
(
  PC$Section  IN Varchar2,   -- CSS section name
  PC$TagName  IN Varchar2,   -- CSS tag name
  PC$TagValue IN Varchar2    -- CSS tag value
);
```

It is used to write/update the value of the given section/tag name

e.g.

```
-- update the value of the tag --
Set_Tag_Value( '.tableHeaderOracle', 'font-family', 'Arial' );
```

- Add a new tag value

```
Procedure Add_Tag_Value
(
  PC$Section  IN Varchar2,   -- CSS section name
  PC$TagName  IN Varchar2,   -- CSS tag name
  PC$TagValue IN Varchar2    -- CSS tag value
);
```

This procedure is used to add a new section/tag value in memory.

- Remove an existing tag

```
Procedure Remove_Tag_Value
(
  PC$Section  IN Varchar2,   -- CSS section name
  PC$TagName  IN Varchar2    -- CSS tag name
);
```

This procedure is used to remove a section/name tag from memory

- Get the complete tag table

```
Function   Get_Tag_Table Return TYP_TAB_CSS ;
```

This function returns a collection of every tags read in the CSS file.
The return type is a collection of records.

- Get all tags of a given section

```
Function   Get_Section_Tags(PC$Section IN Varchar2) Return TYP_TAB_TAG ;
```

This function returns a collection of tags

- Set all tags of a given section

```
Procedure  Set_Section_Tags
(
  PC$Section IN Varchar2,     -- CSS section name
  PT$TTags   IN TYP_TAB_TAG   -- Array of tags found
 ) ;
```

This procedure set every tags of the given section.

- Displaying of an error message

```
Procedure ShowError( PC$Message IN Varchar2 ) ;
```

This procedure is used internally to display an error message. It uses the LAF_AL_ERROR Alert, present in the LAF_TEMPLATE.fmb file and the laf.olb object groups.

- Colour the records of a table-block

```
Procedure Fill_table( PC$Type  IN Varchar2 DEFAULT 'ODD' ) ;
```

This procedure is used to colour the records in the current block.
It is generally called in the *Post-Query* and *When-New-Record-Instance* block-level triggers.

PC$Type can be one of the following:

- **ODD** every odd record are painted
- **EVEN** every even record are painted
- **ALL** every records are painted

The visual attribute used to paint the record is the one given in the Paint_Block() method.

- Get a specific token from a delimited string

```
Function Split
(
  PC$Chaine IN VARCHAR2,          -- input string
  PN$Pos IN PLS_INTEGER,          -- token number
  PC$Sep IN VARCHAR2 DEFAULT ',' -- separator character
) Return Varchar2 ;
```

This function returns the nth token in a delimited string.
Given the original string is: 'one,two,free,four' and you want to get the second element, proceed as follows:

```
LC$Value := Split( 'one,two,free,four', 2 ) ;
```

The function return NULL when the token is not found.
To get every token, use the following code snippet:

```
Declare
  LC$Value  Varchar2(100);
  LN$I      Pls_Integer := 1 ;
Begin
  Loop
    LC$Value := Split( 'one,two,free,four', LN$I ) ;
    Exit When LC$Value Is Null ;
    ...
    LN$I := LN$I + 1 ;
  End loop;
End;
```

If the separator character is not a comma, give it as the third argument.

```
LC$Value := Split( 'one|two|free|four', 2, '|' ) ;
```

- Convert a delimited string to a collection

```
Procedure To_String_Collection
(
   LC$String      IN Varchar2,
   PT$StringTable IN OUT NOCOPY TYP_TAB_STRINGS
 );
```

Description of the functions stored in the PKG_TOOL package:

These functions are used to establish a correspondence between the several coordinate systems that Forms can handle.
As the methods stored in the Java Bean use only the pixel coordinate system, these functions help you to convert the current Forms value to pixel equivalent and vice-versa.

- Get the pixel value corresponding to the current coordinate system given value

```
-- return pixels from any coordinate system --
Function To_Pixel( PN$Coord1  In Number )
Return Pls_Integer ;
```

- Get the current coordinate system value from a pixel given value

```
-- return current coordinate value from pixel value --
Function To_Current_Coord( PN$Coord1  In Number )
Return Number ;
```

- Get the value of two pixel values in a delimited string

```
Function To_Pixel
  (
     PN$Coord1     In Number,
     PN$Coord2     In Number,
     PC$Separator  In Varchar2 Default '|'
   )
Return Varchar2 ;
```

- Get the value of two current coordinate system values in a delimited string

```
Function To_Current_Coord
  (
     PN$Coord1     In Number,
     PN$Coord2     In Number,
     PC$Separator  In Varchar2 Default '|'
   )
Return Varchar2 ;
```

- Init the blocks in order to use the Set_Custom_Property()

```
PROCEDURE init_laf_blocks
(
   PC$Blk1   in varchar2 default null
  ,PC$Blk2   in varchar2 default null
  ,PC$Blk3   in varchar2 default null
) ;
```

It is used to display the blocks located on non-visible canvas, in order to use the Set_Custom_Property() on initialized elements. This procedure has to be added to the When-New-Form-Instance trigger. You can pass one up to three blocknames that you don't want to process, like the LAF or the Webutil blocks.

- Populate the clipboard from a table-block content.

```
PROCEDURE Copy_From_block
(
   PC$Block    in varchar2
  ,PC$Bean     in varchar2
  ,PB$Header   in boolean default FALSE
  ,PN$From     in pls_integer default 1
  ,PC$Items    in varchar2 default null
  ,PC$FieldSep in varchar2 default CHR(9)
) ;
```

It is used to copy a table-block content to the clipboard. The data is exported as ASCII delimited text. You can choose the column you want to export, and also if you want to export the column header.
Only the first two parameters are mandatory: the block name and the Bean Area name.
The PB$Header indicates if you want to export the column header.
The PN$From indicates from what column you want to export. For example, if you don't want to export the first column, pass 2 as this parameter.
The PC$Items can be a comma delimited string that includes the list of columns you want to export.
PC$FieldSep is the character used to separate the columns.

e.g.

```
-- export all columns plus header --
Pkg_Tools.Copy_from_block('USR_TABLES','LAF_BLOCK.LAF_BEAN');
-- export 3 columns without header -
Pkg_Tools.Copy_from_block('USR_TABLES','LAF_BLOCK.LAF_BEAN',false,1,
                          ,'EMPNO,SAL,COMM');
```

See the test_laf_copy_paste_block.fmb that is part of the LAF zip file.

- Populate the table-block from the clipboard.

```
PROCEDURE Paste_to_block
(
   PC$Block    in varchar2
  ,PC$Bean     in varchar2
  ,PB$Header   in boolean default FALSE
  ,PN$From     in pls_integer default 1
  ,PC$Items    in varchar2 default null
  ,PC$FieldSep in varchar2 default CHR(9)
) ;
```

It is used to populate a table-block content from the clipboard content.
The records at created at current record position in the block.
Only the first two parameters are mandatory: the block name and the Bean Area name.
The PB$Header indicates if you want to export the column header.
The PN$From indicates from what column you want to export. For example, if you don't want to export the first column, pass 2 as this parameter.
The PC$Items can be a comma delimited string that includes the list of columns you want to export.
PC$FieldSep is the character used to separate the columns.

You need to fetch the clipboard current content first.

e.g.

```
-- get the clipboard content --
Set_Custom_Property( 'LAF_BLOCK.LAF_BEAN', 1, 'PASTE_FROM_CLIPBOARD', '');

-- paste all columns --
Pkg_Tools.Paste_to_block('USR_TABLES','LAF_BLOCK.LAF_BEAN');
-- paste 3 columns without header -
Pkg_Tools.Paste_to_block ('USR_TABLES','LAF_BLOCK.LAF_BEAN',false,1,
                          ,'EMPNO,SAL,COMM');
```

See the test_laf_copy_paste_block.fmb that is part of the LAF zip file.

- Highlight a record with gradient background.

```
PROCEDURE highlight_record( PC$Block in varchar2 ) ;
```

It is used to highlight the current record when its Items Implementation class is set to oracle.forms.fd.LAF_XP_TextField or oracle.forms.fd.LAF_XP_TextArea, and a gradient background is given via the SET_GRADIENT method.
Call this procedure from the block's When-New-Record-Instance trigger:

```
PKG_TOOLS.highlight_record(:system.current_block);
```

See the test_laf_gradient_fields.fmb that is part of the LAF zip file.

The table-block multi select record feature

It allows the end-user to select/unselect records in a table-block.

When a record is selected, its visual properties are updated to render the selection to the screen.
It uses a Visual Attribute, and its properties can be read from the CSS file

```
multi-select:VA_LAF_MTSELECT,Tahoma,I,10,r0g185b90,r255g255b150
multi-select-modifier:Ctrl
```

Those properties must be defined in a section of GUI type.

multi-select tag defines the Visual Attribute and its properties used to colourize the selected records.

```
va_name[,font_name[,font_weight[,font_size[,foreground[,background]]]]
```

font_weight can be:

- P (plain)
- B (bold)
- I italic)
- PI (plain+italic)
- BI (bold+italic)

If you don't provide all element values, put a minus (-) instead.

```
multi-select:VA_LAF_MTSELECT,Tahoma,-,10,-,r255g255b150
```

As this tag indicates the Visual Attribute used, it must exist at runtime in the Forms module.
If all properties are already defined in this VA, you don't need to provide them in the tag:

```
multi-select:VA_LAF_MTSELECT
```

multi-select-modifier tag defines what keyboard modifier use to select the record in conjunction with the mouse.

possible values are:

- - (none)
- Shift
- Ctrl
- Alt
- Shift+ctrl

If not provided, the tag default is nothing.

The Forms triggers

In order to use the multi-select feature in your table-block, you have to add some code in the following triggers:

form-level:

POST-FORM

```
-- clear the collection before exiting --
pkg_multiselect.clear_all_blocks;
```

This clear the memory used by the module before exiting

block-level:

POST-QUERY

```
-- set initial value to unchecked --
pkg_multiselect.set_state(get_block_property(:system.trigger_block,
CURRENT_RECORD), 0);
```

It creates one in-memory collection element to handle the current record status (selected/unselected). Initial value is 0 (zero) that means unselected.

KEY-EXEQRY

```
-- clear the collection --
pkg_multiselect.clear;
-- execute the query --
execute_query;
```

It clears the in-memory collection before querying the data

KEY-DELREC

```
pkg_multiselect.delete_record(get_block_property(:system.trigger_block,
CURRENT_RECORD));
delete_record;
```

Used to synchronize the in-memory collection while deleting a record

WHEN-CREATE-RECORD

```
pkg_multiselect.create_record(get_block_property(:system.trigger_block,
CURRENT_RECORD));
```

Used to synchronize the in-memory collection while inserting a record

WHEN-MOUSE-[DOUBLE]CLICK

```
pkg_multiselect.change_state(:system.cursor_record);
```

That really do the select/unselect job

Those triggers are grouped in an Object Group in the laf.olb Object Library
The group name is : GRP_MULTISELECT

Drop this group name in your current module then drag the triggers in your final block.

Get the selected record list

At the moment you want to get the selected record list, use the laf.pll *pkg_multiselect.get_checked_list()* function:

```
Declare
      t pkg_multiselect.TAB_SEL;
Begin
      t := pkg_multiselect.get_checked_list('EMP');
      if t.count > 0 then
        for i in 1 .. t.last loop
            message('selected record:' || t(i));
        end loop;
      else
        message('no record selected') ;
      end if ;
End;
```

pkg_multiselect.TAB_SEL is a PL/SQL table of PLS_INTEGERs.

At any time, within the current record, you can know its state (selected/unselected) by using the *pkg_multiselect.get_state()* function

```
Begin
  first_record;
  loop
    message('rec:' || :SYSTEM.CURSOR_RECORD || ' ->'
                   || pkg_multiselect.get_state(:SYSTEM.CURSOR_RECORD));
    exit when :system.last_record = 'TRUE';
    next_record;
  end loop;
End;
```

The function returns 1 for selected record and 0 for unselected record.

The Java Beans and the Pluggable Java Components

The DrawLaf Java Bean

All the graphic operations not in relation with a specific Forms item are done through the methods included in the **DrawLaf** Java Bean.

It permits to manage the following aspects:

- Drawing shapes on the current canvas (images, lines, rectangles and strings).
- Loading and playing sounds.
- Dynamically handling menus - add, remove, enable, disable, show and hide menu options at runtime.
- Dynamically handling frames – add, remove, modify, move, show and hide frames at runtime.
- Display single or multi-line input dialogue box.
- Transform the Forms into a Socket Server, able to receive external asynchronous messages.
- Display texts anywhere on the canvas during a given time.
- Change fonts and colours for menu bar, window caption, status bar and tabs.
- Pick a colour from a JColorChooser.
- Turn simple images into sensitive areas you can click on.
- Display an HTML Scrolling Panel to present large information in a small room.
- Execute orders contained in an external file (Robot feature)
- Create dynamically items at runtime (buttons, textfields, checkboxes and images)
- Handle dynamic table-blocks through Java Jtables.

All its methods are grouped in the oracle.form.fd.DrawLAF Java class.

In order to call these methods, you have to add a Bean Area on the canvas, then set its **Implementation Class** property to:

oracle.forms.fd.DrawLAF

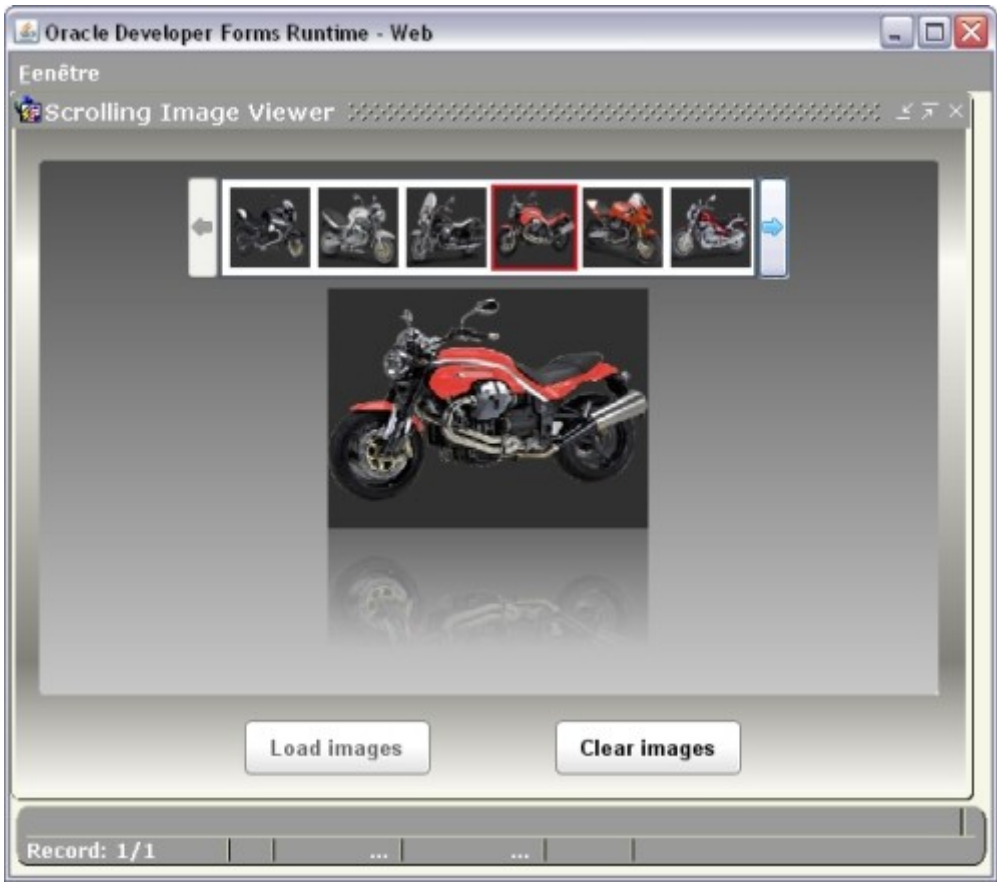
This class is stored in a JAR file, whose name depends on the version of Forms you use:

- **laf_902.jar** if you use the 9.0.2 or 9.0.4 version
- **laf_1012** if you use the 10.1.2 version
- **laf_10123** if you use the 10.1.2.3 version
- **laf_111112** if you use the 11.1.1.2 version

Note:
As you can only draw on the canvas that supports the Bean Area, you need as many Bean Areas as you have different canvases in your Forms application.

Doc: To see the complete list of available methods on this bean, read the DrawLAF Java Bean documentation.

The ImageViewer Java Bean



It is a useful tool to show image collections like photo albums or commercial catalogues.

You can attach it to four different locations from the bean area:

- NORTH (current screen shot)
- SOUTH
- EST
- WEST

It offers more than 20 methods to set-up and display your images in a scrolling bar. Each small icon can display an HTML tool tip, and the main image will send a message back to Forms when you click it, allowing the developer to attach any functions of his own to the image.

This feature needs its own screen area to display the image viewer, so that you have to add another Bean Area to your canvas with the following **Implementation Class** property:

oracle.forms.fd.ImageViewer

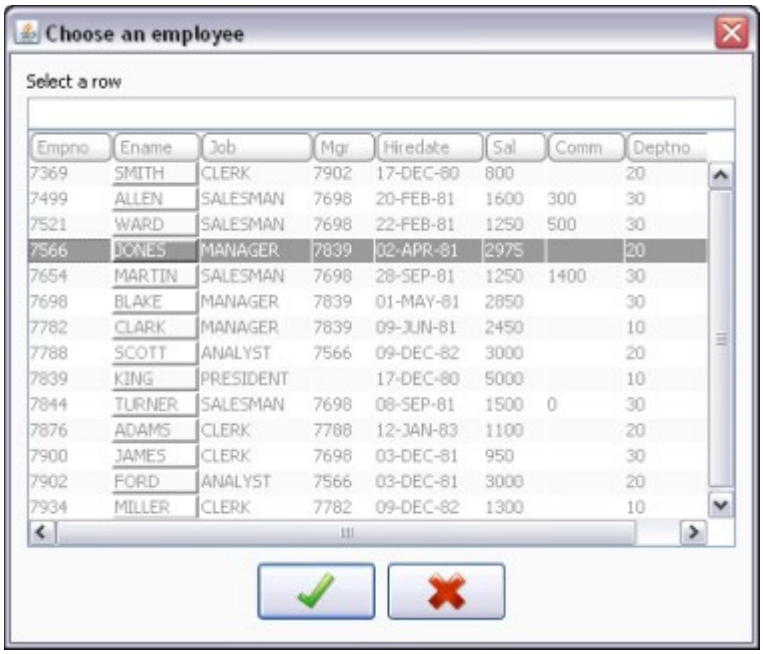
To test it, you would find the **test_laf_image_viewer.fmb** sample dialogue in the **/fmb** folder.
It also uses the **/fmb/icons** folder that contains the images.
Without any modification, the sample dialogue expects to find this folder in the c:/ root. If you want to copy the /icons folder anywhere else, indicate it in the *When-New-Form-Instance* trigger:

```
:GLOBAL.IMAGE_DIR := 'c:/other_place/icons/' ;
```

Doc: To see the complete list of available methods on this bean, read the Carousel Java Bean properties.

The LAF_LOV Java Bean

It allows the developer to show a List of Values (LOV) in a Swing JTable object. It is available since the 1.3.9 version.



It needs a database package to manage the data communication between the database and the Java Bean.
The script of this package is located in the /script folder of the LAF zipped file.

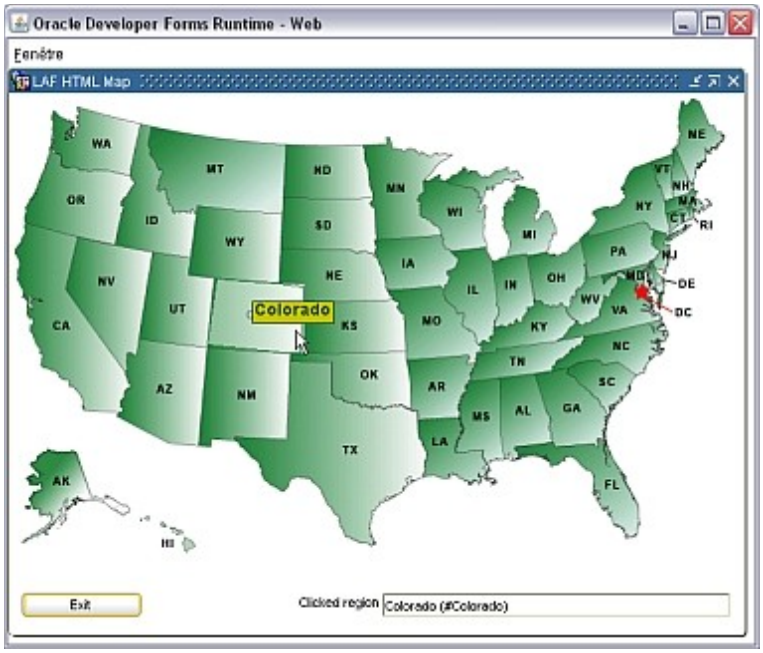
The LOV is decorated in the same way as the table-blocks, and supports the following features:

- Any column can be chosen by the end-user to filter the list.
- Any column can be sorted on.
- Any column can validate any item in the module.
- The end-user can move and resize any column.
- The column mapping permits to map the selected column to more than one return item.
- You can define the LOV buttons' label and even put icons on them.

Doc: To see the complete list of available methods on this bean, read the LAF LOV Bean documentation.

The LAF_Map Java Bean

It allows the developer to handle a HTML Map.
It is available since the 1.6.9 version.



As any HTML map, it needs an image and zone coordinates to describe the different Map areas.

When a Map zone is clicked, a message is sent back to the forms module via the *Set_Custom_Item_Event* trigger associated to the Bean Area.

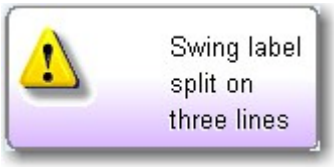
The Implementation Class of the Bean Area must be : oracle.forms.fd.LAF_Map

Doc: To see the complete list of available methods on this bean, read the LAF_Map Bean documentation.

The Pluggable Java Components (PJCs)

Some of the Standard Forms Widgets can be overloaded to change their look and extend their functionalities.

- Push Button



The Implementation Class property needed to overload a standard Push Button is : **oracle.forms.fd.LAF_XP_Button**

Doc: To see the complete list of available methods on this PJC, read the LAF_XP_Button properties documentation.

- Check-box

The Implementation Class property needed to overload a standard Check box is : **oracle.forms.fd.LAF_XP_CBox**

Doc: To see the complete list of available methods on this PJC, read the LAF_CheckBox properties documentation.

- Radio Button

The Implementation Class property needed to overload a standard Radio Button is : **oracle.forms.fd.LAF_XP_RadioButton**

Doc: To see the complete list of available methods on this PJC, read the LAF_XP_Button properties documentation.

- Single-line Text Item

The Implementation Class property needed to overload a standard single-line Text Item is : **oracle.forms.fd.LAF_XP_TextField**

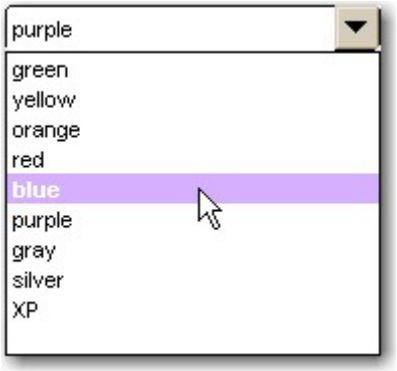
Doc: To see the complete list of available methods on this PJC, read the LAF_XP_TextField properties documentation.

- Multi-line Text Item

The Implementation Class property needed to overload a standard multi-line Text Item is : **oracle.forms.fd.LAF_XP_TextArea**

Doc: To see the complete list of available methods on this PJC, read the LAF_XP_TextArea properties documentation.

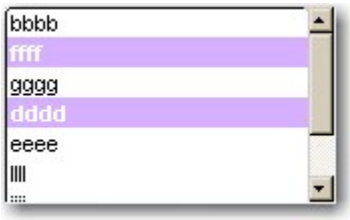
- Poplist item



The Implementation Class property needed to overload a Poplist Item is : **oracle.forms.fd.LAF_XP_PopList**

Doc: To see the complete list of available methods on this PJC, read the LAF_XP_PopList properties documentation.

- Tlist item



The Implementation Class property needed to overload a Tlist Item is :
oracle.forms.fd.LAF_XP_TList

Doc: To see the complete list of available methods on this PJC, read the LAF_XP_TList properties documentation.

Implementation in the Forms modules

- Using the Template form

  In the **/fmb** folder of the zip file, you would find a template named : **LAF_TEMPLATE.fmb**.

  While you are building a brand new module from scratch, it is best to create the new module from this template, by using the **File → New → Forms using template...** Forms Builder menu option.

  Every component needed to use the LAF features will be incorporated.

- Using the Objects Library

  You can also drag the GRP_LAF **laf.olb** Object Library's group to the Objects Groups node of an empty Forms module.

- Update of existing modules

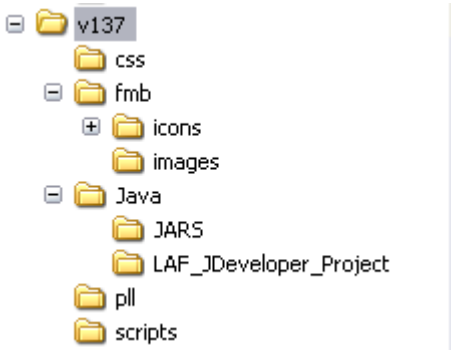
  While you want to update existing modules, you have to use the **LAF_JDAPI** tool.

  It is made of a JAR file and a XML configuration file.
  The XML file is used to indicate the list of the modules you want to update, allowing you to update many modules in a one shot.

  See the JDAPI_LAF tool

Download the LAF Project zip file

You can download the last version from the Look and Feel Project home page.
Here is a description of the content of the zip file:



- The **/css** sub-folder contains the current **forms.css** template CSS file.

  Without any modification, this file is generally expected in the c:/ root directory.

  There are several places you can indicate the location of this file.

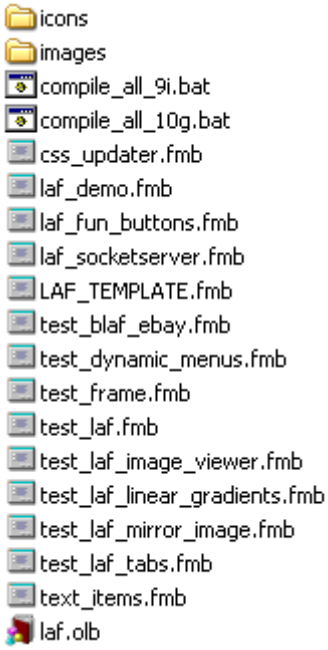
  The *PM$CSS_FILENAME* Forms parameter is one of them.

  If you create a new module from the **LAF_TEMPLATE.fmb** file or if you use the GRP_LAF **laf.olb** Object Library group, it would be present in your module.

  You can also indicate the full path directly in the *PKG_Look_And_Feel.Open_Css()* laf.pll's function.

  If you are updating existing Forms module via the LAF_JDAPI tool, you can also indicate the location of the CSS file in the XML configuration file.

- The **/fmb** sub-folder contains the Forms sample dialogues, the Object Library, the demo icons and images, and two batch files to compile the modules.



The **laf_demo.fmb** sample dialogue is a starting screen that groups almost all the other demos.

Module compilation

Since, you have decompressed the zip file, and copy the Forms samples and the PL/SQL library, you have to compile the modules.

Two batch files are provided to achieve that task:

compile_all_9i.bat and **compile_all_10g.bat** depending on the Forms version you use. (*Use compile_all_10g for Forms 11g*)

The only argument to pass to these batch files is the Database connection string.

Assume yours is : test/test@xe, and you are using Forms 10g, compile the modules with the following:

```
compile_all_10g.bat test/test@xe
```

Since the **laf.pll** has been compiled, move a copy of both **laf.pll** and **laf.plx** files in one of the folders pointed by the **FORMS_PATH** environment variable.

- The **/Java** folder contains two sub-folders:

  /JARS that contains 4 JAR files:

  - laf_902.jar
  - laf_1012.jar
  - laf_10123.jar
  - laf_11112.jar

  Only one of them is to copy to your **/forms/java** directory, depending of the Forms version used.

  /LAF_JDeveloper_Project that contains the whole Oracle Jdeveloper project (Jdeveloper 10.1.3.1), for you, to adapt, enhance or simply rebuild from Oracle Jdeveloper.

- The **/pll** folder that contains the **laf.pll** PL/SQL library.

  It is required to attach this library to your Forms modules, if you want to use the CSS feature like painting the canvas or the blocks.

  It is not required to attach this library if you do not want to use the CSS features, like decorating the blocks or tuning the general GUI settings.

  Actually, almost every CSS tag feature has an equivalent *Set_Custom_Property()* associated method, that you can call "manually" from anywhere in the Forms code.

- The **/scripts** folder contains some scripts to maintain the Database objects.

  PKG_LAF.sql

  This PL/SQL package is needed to transfer LOB chunks between the Database and the Java Bean.

  It is particularly used by the *Read_Image_Base()* ImageViewer's Bean and the *Set_Sound_Base()* DrawLAF's methods.

  PKG_DB_LAF_LOV.sql

  This PL/SQL package is needed to use the Swing JTable LOVs.

Examples

- **A basic starting PL/SQL code**

Here is a basic PL/SQL code you would use when the forms starts:

```
-------------------------------------
--    form main initializations    --
-------------------------------------

If PKG_Look_And_Feel.Open_Css(:PARAMETER.PM$CSS_FILENAME) Then

  -- read the global GUI properties --
  PKG_LOOK_AND_FEEL.Set_GUI_Properties( '.GUIProperties1',
                                        'LAF_BLOCK.LAF_BEAN' ) ;

  -- paint the canvases --
  PKG_LOOK_AND_FEEL.Paint_Canevas(:PARAMETER.PM$CANVAS,
                                  'LAF_BLOCK.LAF_BEAN' ) ;

  -- paint the blocks --
  PKG_LOOK_AND_FEEL.Paint_Block
    (
      PC$Block      => 'EMP'
     ,PC$BeanName   => 'LAF_BLOCK.LAF_BEAN'
     ,PC$VA_Name    => :PARAMETER.PM$VA
     ,PC$HeadClass  => :PARAMETER.PM$HEADER
     ,PC$BodyClass  => :PARAMETER.PM$BODY
     ,PC$TitleClass => :PARAMETER.PM$TITLE
     ,PC$Title      => 'Theme #1 for this table block'
     ,PB$ScrollBar  => True
    ) ;

End if ;
```

The CSS file is loaded in memory, then the GUI properties are setted, finally, the main canvas and the table-block(s) are painted.

Notice, that, if you have more that one block on the canvas that supports the bean, you have to call the *Paint_Block()* procedure for each block.

Warning:

Because a Forms Bean Area supports a Java Bean, it has to be initialized before you can use its methods. It is the reason why it is not recommended to use the *Set_Custom_Property()* and *Get_Custom_Property()* built-ins in the very starting phases of a Forms module life, and those starting phases include the *When-New-Form-Instance* and *New-Block-Instance* triggers.

The common tip, generally, is to introduce a short delay in the *When-New-Form-Instance* trigger.

For this purpose, you can use, at least, two different methods:

- **Use a timer**

  All you have to do is to create a non-repeating timer, then move the specific LAF code to the *When-Timer-Expired* trigger:

  When-New-Form-Instance trigger:

```
Declare
   timer_id   Timer ;
Begin
  --  need a while before beans are initialized --
  timer_id := Create_Timer( 'laf_timer', 50, NO_REPEAT ) ;
End ;
```

  When-Timer-Expired trigger:

```
If lower(Get_Application_Property( TIMER_NAME )) = 'laf_timer' Then


  -------------------------------------
  --    form main initializations    --
  -------------------------------------
  If PKG_Look_And_Feel.Open_Css(:PARAMETER.PM$CSS_FILENAME) Then
     ...
  End if ;


End if;
```

- **Use the *DBMS_LOCK.Sleep* Database procedure**

  Another solution, when you are sure that the Forms module is connected to the Database, is to introduce a short delay by using the *DBMS_LOCK.Sleep()* procedure before setting the custom properties.

  When-New-Form-Instance trigger:

```
dbms_lock.sleep(2/10);
If PKG_Look_And_Feel.Open_Css(:PARAMETER.PM$CSS_FILENAME) Then
  ...
End if ;
```

  This method is highly recommended as it avoids the "flashing screen" disappointment.

- **Special spread-table block setting.**

  While you have one block spread on two canvases, as it is the case when you build a spread table, where some items are located on the main canvas, and other items spread on a stacked canvas, you have to call the *Paint_Block()* twice, once with the name of the Bean Area located on the main canvas, and once with the name of the Bean Area located on the stacked canvas.

- **Multi-canvases module.**

  If the module contains more than one canvas, and you want to use the LAF features on each of them, you need to put a Bean Area on, at least, one block for each different canvas.

  Warning:

  You cannot use the methods of a Java Bean while it has not been initialized, and a Java Bean is initialized only when it is on a visible canvas, and this canvas is displayed.

  In short, you cannot use the *Set_Custom_Property()* and *Get_Custom_Property()* built-ins with a Java Bean located on a canvas that has never been displayed.

  To avoid a Java runtime error that will freeze the Java Bean, you have to display every canvas that supports a Bean Area with the *Show_View()* built-in while the Forms is started (*When-New-Form-Instance* or *When-Timer-Expired* triggers).

Here is the PL/SQL code you would write for a module that contains two canvases:

```
If PKG_Look_And_Feel.Open_Css(:PARAMETER.PM$CSS_FILENAME) Then

    -- paint the canvas --
    PKG_LOOK_AND_FEEL.Paint_Canevas(:PARAMETER.PM$CANVAS, 'CTRL.BEAN' ) ;
    -- set the GUI properties --
    PKG_LOOK_AND_FEEL.Set_GUI_Properties( '.GUIProperties1', 'CTRL.BEAN' ) ;

    -- paint the blocks --
    PKG_LOOK_AND_FEEL.Paint_Block
    (
      PC$Block      => 'EMP'
     ,PC$BeanName   => 'CTRL.BEAN'
     ,PC$VA_Name    => :PARAMETER.PM$VA
     ,PC$HeadClass  => :PARAMETER.PM$HEADER
     ,PC$BodyClass  => :PARAMETER.PM$BODY
     ,PC$TitleClass => :PARAMETER.PM$TITLE
     ,PC$Title      => 'Theme #1 for this table block'
     ,PB$ScrollBar  => true
    ) ;

    Go_Block('EMP');
    -- populate the block --
    Execute_Query ;

  --
  -- hidden canvases that supports PJCs must be displayed once
  -- to initialize the bean areas and PJCs implementation classes
  --
  Show_View('CV2');

  -- set some individual properties --
  PKG_LOOK_AND_FEEL.Paint_Canevas('.canvasBlue2', 'CTRL.LAF' ) ;

  ...

  -- come back to first canvas/block --
  Go_Block('EMP');

End if ;
```

Here is a procedure you can call from the When-Timer-Expired trigger, that does the job:

```
PROCEDURE init_laf_blocks
(
   PC$Blk1   in varchar2 default null
  ,PC$Blk2   in varchar2 default null
  ,PC$Blk3   in varchar2 default null
)
IS
      LC$blockDeb  varchar2(60); -- start block
      LC$block     varchar2(60); -- current block name
      LC$item      varchar2(60); -- current item
      LC$itemdeb   varchar2(60); -- first item
BEGIN
      LC$BlockDeb := get_form_property( NAME_IN('System.Current_Form')
                                          , FIRST_BLOCK ) ;
      LC$Block := LC$BlockDeb ;
      Loop -- For each block of the form
            If  LC$Block != Upper( Nvl( PC$Blk1, ' '))
            And LC$Block != Upper( Nvl( PC$Blk2, ' '))
            And LC$Block != Upper( Nvl( PC$Blk3, ' '))  Then
                  LC$itemdeb := get_block_property(LC$BLOCK, FIRST_ITEM) ;
                  LC$item := LC$BLOCK || '.' || LC$itemdeb ;
                  While LC$itemdeb is not null loop -- For each item
                        -- navigable item ? -
                        IF GET_ITEM_PROPERTY(LC$Item , NAVIGABLE) = 'TRUE' Then
                              Go_Block(LC$Block);
                              Synchronize;
                              exit;
                        END IF;
                        LC$itemdeb := get_item_property(LC$item, NEXTITEM );
                        LC$item := LC$BLOCK || '.' || LC$itemdeb ;
                  End loop ;
            End if ;
            LC$Block := get_block_property( LC$Block, NEXTBLOCK ) ;
            exit when LC$Block is null ;
      End loop ;
END init_laf_blocks;
```

This procedure is part of the **laf.pll** PL/SQL library in the **PKG_TOOLS** package.
It accepts up to 3 arguments, that define block you don't want to proceed.

When-Timer-Expired trigger:

```
...
 PKG_TOOLS.init_laf_blocks('LAF_BLOCK', 'WEBUTIL');
 -- there, you can use the Set_Custom_Property() on every block/item.
...
```

Acknowledgements

This tool would probably not exist without the support of Grant Ronald, who, first, gave me the idea, that introduced the need.

Create something is always great, but without the original idea, nothing can exists ;o)

Many thanks to the people from the Oracle Forms Managers Grant Ronald, Frank Nimphius and Duncan Mills for their support all along the past years.

Special thanks to the Oracle Forms Development Team in general who created and maintained this fabulous product, allowing the developer to mix with as many Java code as we want.

Special thanks for the people that tried, used, tested, raised bugs and also provided code snippets and enhancements.

Developer list

This tool has been developed by the following people:

- Francois Degrelle (creator)
- John Vander Heyden (contributions in the laf.pll)
- Albert Ellen (contributions in the laf.pll)
- Tom Cleymans (for the use of his DispatchingBean solution)
- Anthony Hegarty (contribution in the DrawLAF.java)
- BUI Thanh Hoang (contribution in laf.pll and DrawLAF.java)

Oracle Forms Look & Feel project

Created and maintained by Francois Degrelle

Oracle Forms L&F Web site