GENERIC PROGRAMMING IN ORACLE FORMS

Rune Mørk, NNE A/S

Introduction

As J-developer finally seems to be taking off, now is even more the time to brush up your Oracle Forms skills, as Oracle says in its SOD of June 2004: “forms is here to stay”. As a forms developer you need to be able to compete with J-developer, if you are not yet keen on taking on java skills, and this means that Oracle*forms has to be able to offer more that “just” a productive environment.

Oracle forms offers object-like capability and the possibility to create reusable code, code that are independent of the context it is going to used in. Sadly this possibility is more than often bypassed in the need for development speed. Therefore i will in this article show how you can utilize forms object like capabilities, to create truely reusable code.

In order to utilize these methods you need to learn about object libraries, plsql libraries, the get and set build-ins as well as name-in and copy build-ins.

Definition on generically

Programmers in the Java environment often talk about doing things “the right way” and that other programmers will regard you with contempt when not programming in the true Object Oriented way.

In the forms environment, the tendency is not the same, only in the rare cases where development guidelines has been produced, reuse of code has been seen, but I’ve seen consultancy companies carefully equipping their consultants with a pre prepared package of predefined objects and code to give development a quick start.

Having myself recently embarked on the java transition path I’ve seen many good things in java, methodology wise that forms and pl/sql could benefit from, including the concept of doing the programming the right way, and the inter community mechanism witch is governing the standard of the code produced.

If Forms and plsql development could amend and accept these non-formal guidelines, development of code would, in the longer term, be faster, and maintaining code and forms would be easier.

There is a few other reasons for embarking on the reusable-coding path, one of them is the commonality of the code, writing truly reusable code will mean that the structure and way of coding would be more uniform, things has to be done “the right way”. Another bi-product is the possibility of creating forms with the same look and feel.

Methods in use

In order to be able to reuse your code the development environment needs to be considered carefully, besides that there is a number of technologies you need to master. Therefore I will go through the programming techniques needed, and end up in presenting how such a development environment should be configured. The techniques in play are the following

- Specific coding techniques, by using name_in and copy.
- Knowing the forms objects and it associated get and set options.
- System variables
- Object libraries
- Subclassing

I will describe and exemplify each of these concepts in a simple manner before showing an example in a production environment.

Name_in and Copy

Name in and copy offers unique opportunity to program without referencing directly to forms items.

The function Name_in, see syntax in figure 1, returns the evaluation of en expression in its argument.

```
FUNCTION NAME_IN(variable_name  VARCHAR2) return VARCHAR2;
```

Figure 1 Name_in syntax

For instance referencing to items in a specific block see code in figure 2

```
    v_x := name_in('odtug_master.'||:system.current_item);
```

Figure 2 Getting the value of a specific item

The procedure copy moves the content from source to destination, see syntax in figure 3.

```
PROCEDURE COPY
  (source         VARCHAR2,
   destination    VARCHAR2);
```

Figure 3 Copy syntax

For instance, copying data from one variable to a forms field, as shown in figure 4.

```
COPY(v_data, 'odtug_MASTER.d'||i);
```

Figure 4 Usage of copy

These build-ins are very useful when writing code, where you programmatically are referencing fields in the form dynamically. I've used these build-ins for screen where I was bypassing a master detail relationship that did exist but it was blocking for creation of a user friendly interface. Consider the data-model show in figure 5, witch could symbolize a model of a master and an occurrence during time, e.g. trains on a specific day, seat in a row in a stadium etc, in both examples we know the number of occurrences of the detail in respect to each master.

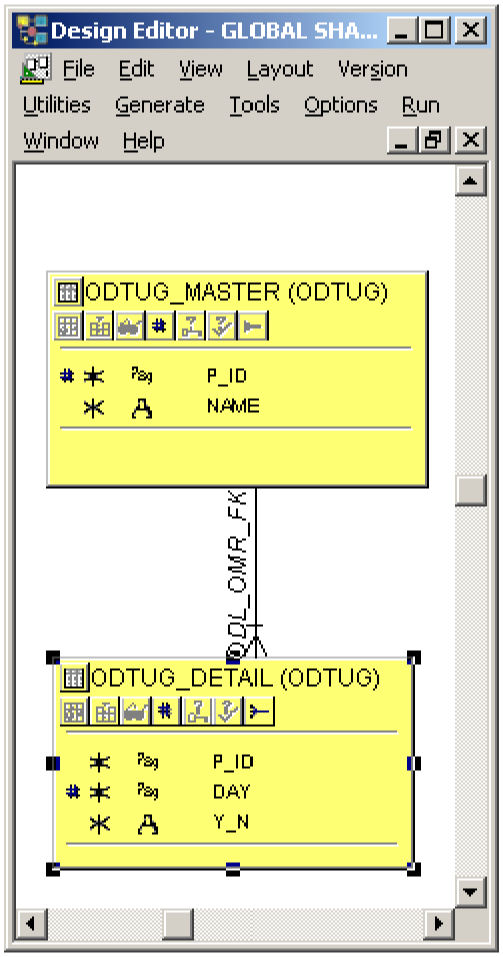
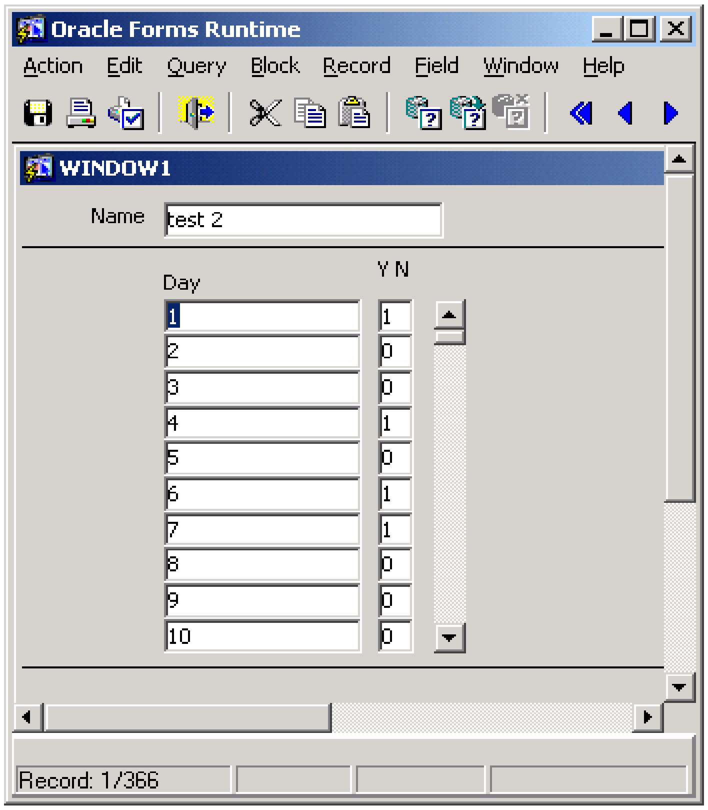

Design Editor - GLOBAL SHA...
File Edit View Layout Version
Utilities Generate Tools Options Run
Window Help
ODTUG_MASTER (ODTUG)
P_ID
NAME
ODL_OMR_FK
ODTUG_DETAIL (ODTUG)
P_ID
DAY
Y_N

Figure 5 Data model

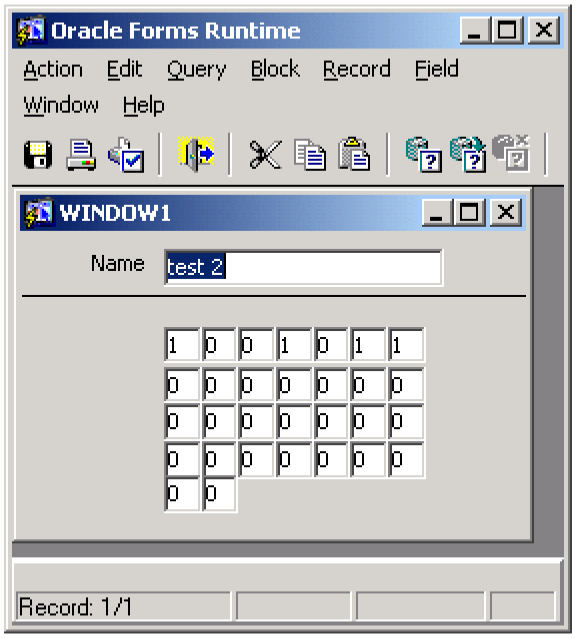
Now in traditional sense you would create a master detail relationship form where you had to navigate from one detail to another in the traditional way, doing so will result in a form looking more or less like the one in figure 6. When utilizing name_in and copy you can get a layout like the one presented in figure 7.



Name	test 2

Day	Y N
1	1
2	0
3	0
4	1
5	0
6	1
7	1
8	0
9	0
10	0

Figure 6 Normal layout



Name	test 2

1	0	0	1	0	1	1
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0					

Figure 7 Enhanced layout

In order to create these forms we had to create generalised code for 2 triggers, the post_query and a block level when validate trigger, see the code for these in figure 8 and 9.

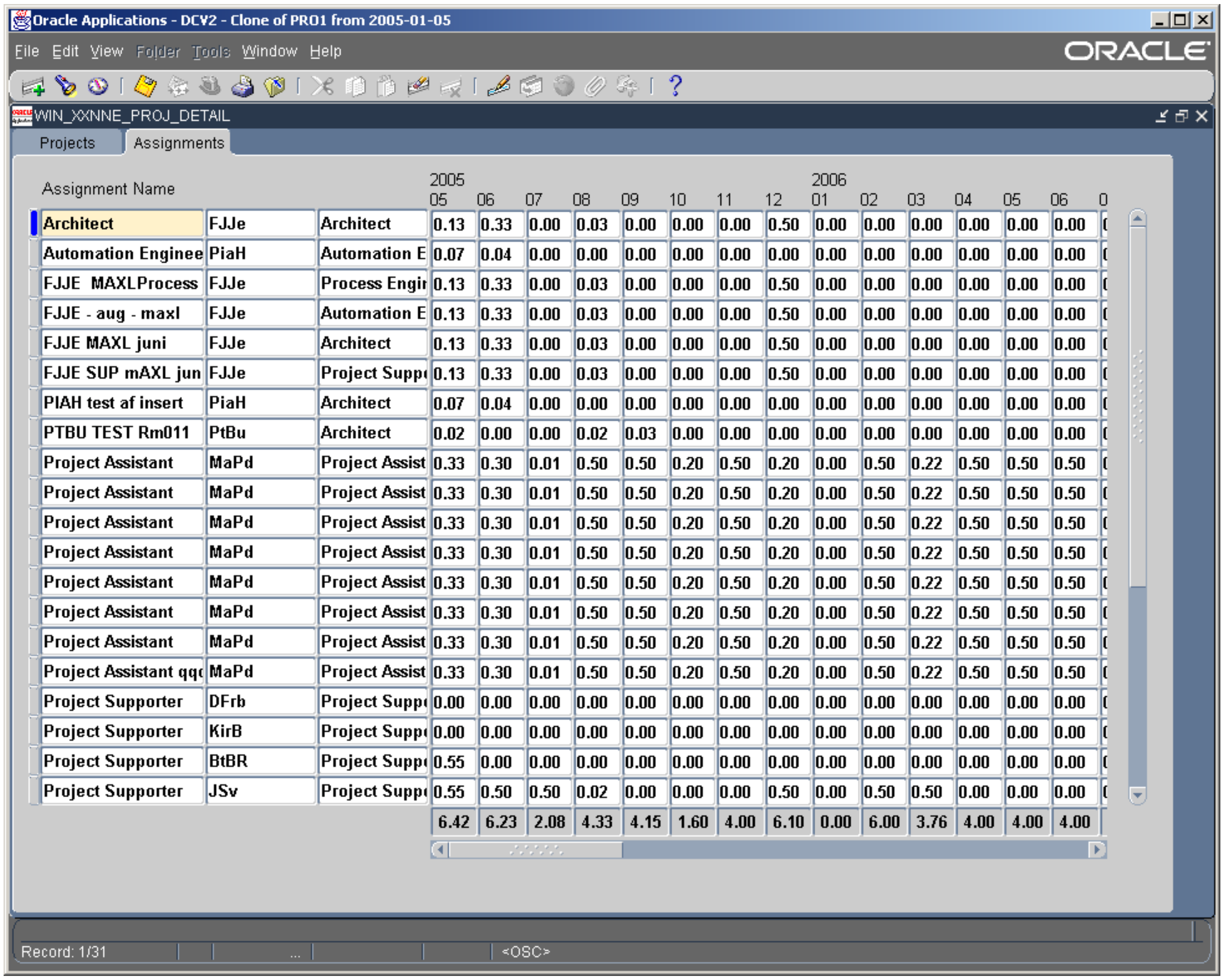
```
declare
 v_data char;
begin
  for i in 1..30 loop
     select Y_N
    into v_data
     from odtug_detail
     where p_id = :odtug_master.p_id
       and day = i;
    copy(v_data, 'odtug_MASTER.d'||i);
  end loop;
end;
```

Figure 8 Post_query

```
declare
 v_x number;
 v_v varchar2(2);
begin
 if :System.Mode = 'NORMAL' then
     if substr(:system.current_item,1,1) = 'D' then
      v_x := name_in('odtug_master.'||:system.current_item);
      v_v := substr(:system.current_item,2,2);
      update ODTUG_DETAIL
      set Y_N = v_x
      where p_id = :odtug_master.p_id
      and day = v_v;
      set_record_property(:system.cursor_record, :system.cursor_block, status, changed_status);
     end if;
 end if;
end;
```

Figure 9 When-validate-item

The screenshot in figure 10 is an example from a real form of how name_in and copy has been utilized. Here a postquery and a when validate item trigger has been created, to maintain data, much more elaborate examples can also be created.



Oracle Applications - DCV2 - Clone of PRO1 from 2005-01-05

WIN_XXNNE_PROJ_DETAIL

Assignment Name			2005								2006						
			05	06	07	08	09	10	11	12	01	02	03	04	05	06	0
Architect	FJJe	Architect	0.13	0.33	0.00	0.03	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00	
Automation Enginee	PiaH	Automation E	0.07	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
FJJE MAXLProcess	FJJe	Process Engi	0.13	0.33	0.00	0.03	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00	
FJJE - aug - maxl	FJJe	Automation E	0.13	0.33	0.00	0.03	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00	
FJJE MAXL juni	FJJe	Architect	0.13	0.33	0.00	0.03	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00	
FJJE SUP mAXL jun	FJJe	Project Supp	0.13	0.33	0.00	0.03	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00	
PIAH test af insert	PiaH	Architect	0.07	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
PTBU TEST Rm011	PtBu	Architect	0.02	0.00	0.00	0.02	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
Project Assistant	MaPd	Project Assist	0.33	0.30	0.01	0.50	0.50	0.20	0.50	0.20	0.00	0.50	0.22	0.50	0.50	0.50	
Project Assistant	MaPd	Project Assist	0.33	0.30	0.01	0.50	0.50	0.20	0.50	0.20	0.00	0.50	0.22	0.50	0.50	0.50	
Project Assistant	MaPd	Project Assist	0.33	0.30	0.01	0.50	0.50	0.20	0.50	0.20	0.00	0.50	0.22	0.50	0.50	0.50	
Project Assistant	MaPd	Project Assist	0.33	0.30	0.01	0.50	0.50	0.20	0.50	0.20	0.00	0.50	0.22	0.50	0.50	0.50	
Project Assistant	MaPd	Project Assist	0.33	0.30	0.01	0.50	0.50	0.20	0.50	0.20	0.00	0.50	0.22	0.50	0.50	0.50	
Project Assistant	MaPd	Project Assist	0.33	0.30	0.01	0.50	0.50	0.20	0.50	0.20	0.00	0.50	0.22	0.50	0.50	0.50	
Project Assistant	MaPd	Project Assist	0.33	0.30	0.01	0.50	0.50	0.20	0.50	0.20	0.00	0.50	0.22	0.50	0.50	0.50	
Project Assistant qq	MaPd	Project Assist	0.33	0.30	0.01	0.50	0.50	0.20	0.50	0.20	0.00	0.50	0.22	0.50	0.50	0.50	
Project Supporter	DFrb	Project Supp	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
Project Supporter	KirB	Project Supp	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
Project Supporter	BtBR	Project Supp	0.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
Project Supporter	JSv	Project Supp	0.55	0.50	0.50	0.02	0.00	0.00	0.00	0.50	0.00	0.50	0.50	0.00	0.00	0.00	
			6.42	6.23	2.08	4.33	4.15	1.60	4.00	6.10	0.00	6.00	3.76	4.00	4.00	4.00	

Record: 1/31 ... <OSC>

Figure 10 A real world example

Get and set

The get and set build-ins are programmatically ways to alter or get a forms-"object"'s properties. Consult the table in figure 11 to see what objects has get and set mechanism attached to them. It will prove way too difficult to explain all the properties that can be read and altered. What you should focus on with these is how to utilise them to alter behaviour and appearance whenever programming.

Object	Get-able	Set-able	Comment
Alert		Yes	Setting info regarding alerts, there is no way of getting
Application	Yes	Yes	Getting info regarding how this are setup in session, some of these properties can only be set programmatically
Block	Yes	Yes	
Canvas	Yes	Yes	
Form	Yes		
Record groups	Yes	Yes	
Item	Yes	Yes	For all items in multi row block
Item_instance	Yes	Yes	Single item instance
List box	Yes		
List of values	Yes	Yes	
Menu	Yes	Yes	
Message	Yes		Getting the current value of the message line (sic!)
Parameters	Yes	Yes	
Radio buttons	Yes	Yes	
Record		Yes	
Relations	Yes	Yes	Getting or setting relation properties, you probably do not want to set these.
Tab_page	Yes	Yes	
Timer		Yes	
Visual attributes	Yes	Yes	
View	Yes	Yes	
Window	Yes	Yes	

Figure 11 Get-able and set-able options

It is crucial when you create reusable code that you master most of these build-ins.

An example of how to utilize the get and set option is by coding generic alerts as shown in figure 12

```
Function manage_alert(p_title in varchar2,
                      p_text in varchar2,
                      p_button1 in varchar2,
                      p_button2 in varchar2 default null,
                      p_button3 in varchar2 default null) return number IS
v_alert alert;
v_return number;
v_alert_name varchar2(10) := 'ALERTB1' ; --  assuming at least one button
BEGIN
  if p_button3 is not null then
  -- this is an alert with 3 buttons
    v_alert_name := 'ALERTB3';
  elsif p_button2 is not null then
  -- this is an alert with 2 buttons
    v_alert_name := 'ALERTB2';
  end if;
  v_alert := find_alert(v_alert_name);
  set_alert_property(v_alert, title, p_title);
  set_alert_property(v_alert, alert_message_text, p_text);
  set_alert_button_property(v_alert, Alert_button1, label, p_button1);
  set_alert_button_property(v_alert, Alert_button2, label, p_button2);
  set_alert_button_property(v_alert, Alert_button3, label, p_button3);
  v_return := show_alert(v_alert);
  RETURN V_RETURN;
 END;
```

Figure 12 Generic alerts

So now you can access this alert by the code shown in figure 13, and for the future you do not have to bother anymore about alerts.

```
declare
 v_ret number;
begin
 v_ret := manage_alert('My Title',
           'My text',
           'OK',
           null,
           null);
end;
```

Figure 13 Using the alert

Another example in figure 14 show shows how alter an item so it looks like disabled, a similar procedure exist to enable an item

```
PROCEDURE DIS_ITEM(P_ITEM IN ITEM, P_REC NUMBER) IS
BEGIN
  if get_item_property(p_item, visible) = 'TRUE' then
    set_item_instance_property(P_ITEM, P_REC, NAVIGABLE, PROPERTY_FALSE);
    set_item_instance_property(P_ITEM, P_REC, UPDATEABLE, PROPERTY_FALSE);
    if get_item_property(p_item,ITEM_TYPE) <> 'CHECKBOX' THEN
      set_item_instance_property(P_ITEM, P_REC, VISUAL_ATTRIBUTE, 'STD_TEXT_ITEM_DISPLAY');
    END IF;
  end if;
END;


PROCEDURE ENA_ITEM(P_ITEM IN ITEM, P_REC IN NUMBER) IS
BEGIN
  if get_item_property(p_item, visible) = 'TRUE' then
    set_item_instance_property(P_ITEM, p_REC, NAVIGABLE, PROPERTY_TRUE);
    set_item_instance_property(P_ITEM, P_REC, UPDATEABLE, PROPERTY_TRUE);
    set_item_instance_property(P_ITEM, P_REC, VISUAL_ATTRIBUTE, 'std_text_item');
 end if;
END;
```

Figure 14 Disabling and enabling items

An example of how this code might influence your layout can be seen in figure 15

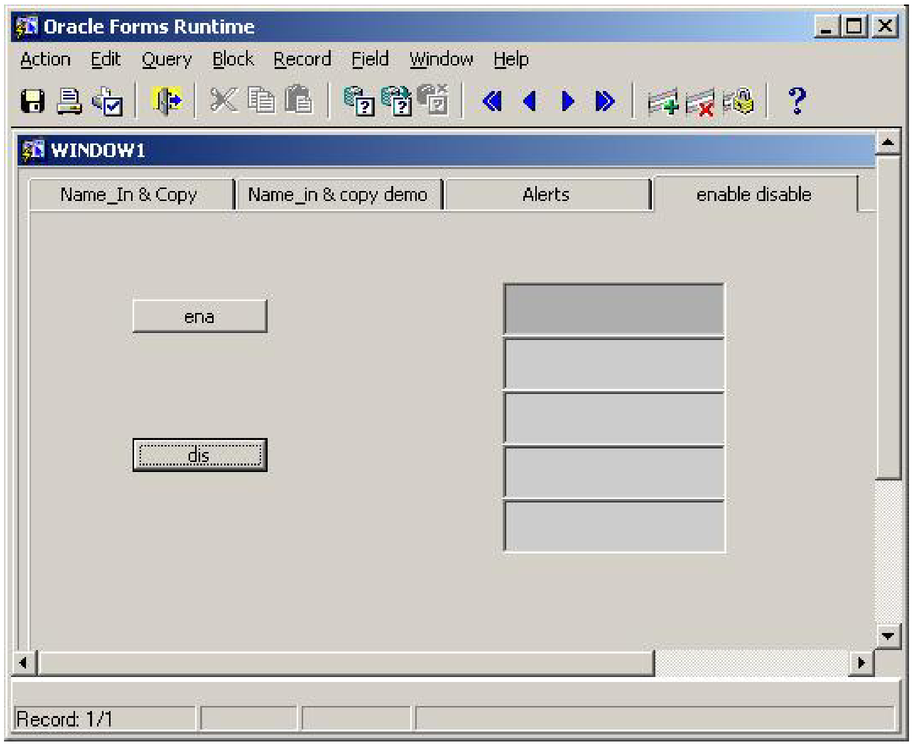


Figure 15 Disabling and enabling items example

In order to fully use the get and set options you also need to familiarize with the system variables. In the example in figure 16 I've utilized the system variables to create a generic trigger that reacts to double click with your mouse.

```
if get_item_property(:system.current_block||'.'||:system.current_item, item_type) = 'TEXT ITEM'
  then
  if get_item_property(:system.current_block||'.'||:system.current_item, list) = 'TRUE' then
    do_key('list_values');
  elsif get_item_property(:system.current_block||'.'||:system.current_item, editor_name) is not null
  then
    do_key('edit_textitem');
  end if;
end if;
```

Figure 16 When mouse double click

So if your current item has a list of value attached, a double click in this will bring up the lov, if the item has en editor attached this editor will be displayed.

Object libraries

In order to create the same look and feel you can utilize object libraries. In an object library you can put all objects you normally can enter into a form. Although you should refrain from putting pl/sql pbject into an object library even though it is possible, instead use a pl/sql-library, as explained later in this article.

In the above example (figure 12) we use three alerts, in order to use these alerts we must create them beforehand, we could do so in each form we creates, or we could include them from an object library. Despite its name it has nothing to do with true object oriented programming, but its merely a canister to place forms-"objects" into, and later reference them to your form. See figure 17 for an example of this.

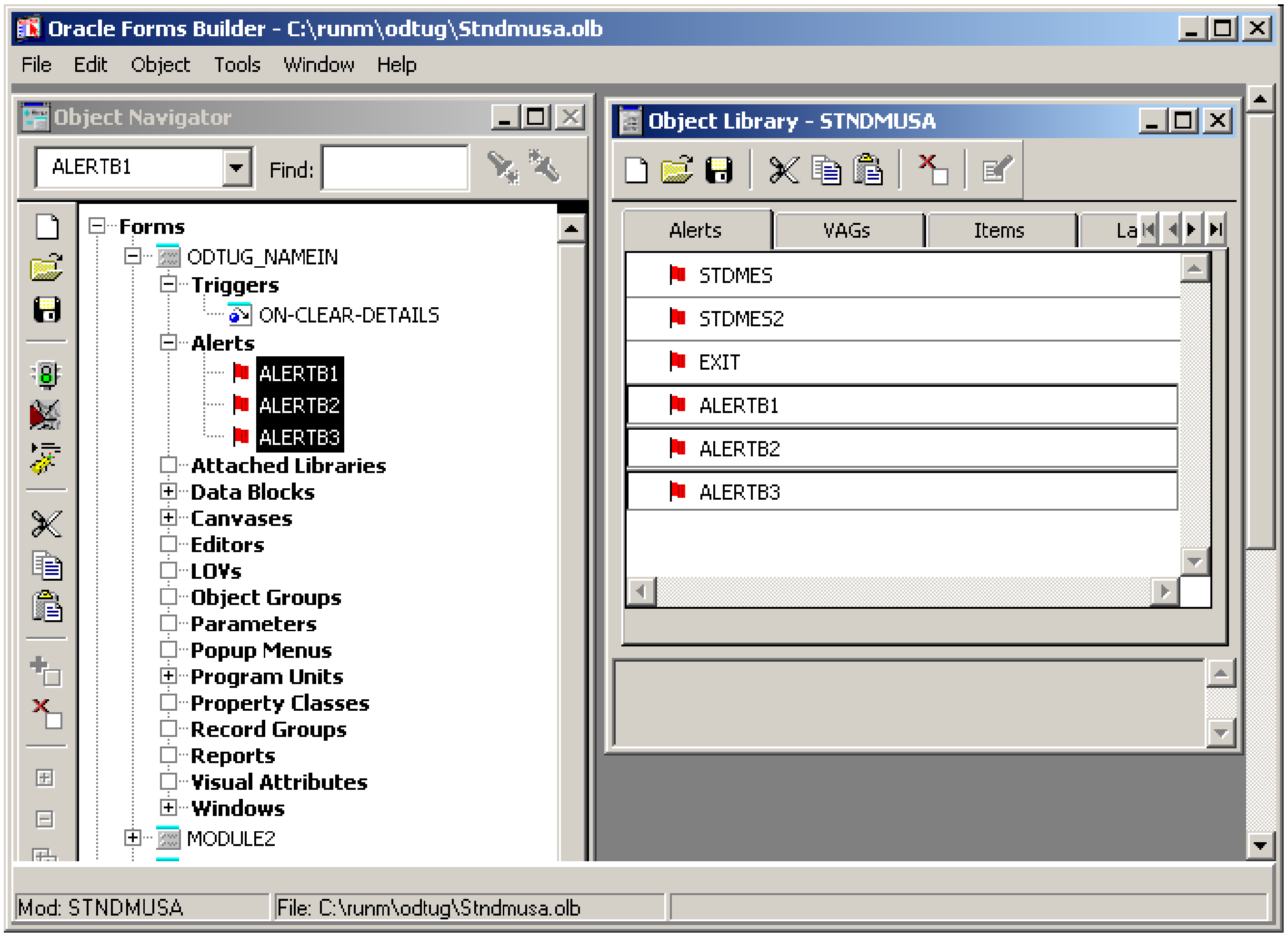


Figure 17 Referencing from Object library.

Sadly it is not possible to edit objects that are placed in an object library; you need to copy them from the object library into a temporary form. To avoid this consider having your reusable object placed in a form and then, whenever changes are necessary, moving the objects into the olb-file. This will give you extra work maintaining this environment, but is worthwhile.
You should consider marking object you place in an object library with a smart class, making them easier accessible from within your form.

It is possible to keep triggers attached to object you put into an object library, but it is not recommendable, in order to access them you need to perform the actions described above, so its time consuming and not quite logical. Instead you should consider using plsql-libries as mentioned in the next section.

Plsql libraries

A plsql-library is the place to save reusable plsql-programs. When you create your programs independently of specific objects then you can put your pl/sql code into a library and compile it.

One thing about pl/sql libraries that always has divided Oracle developers, is to choose between to attach a precompiled library, a *.plx file or keep the normal *.pll file, I cannot judge if one or the other method is right, but based on experience I would keep the file as *.pll. I've seen many situations where developers forgot to compile their library, and therefore they were unable to have their changes take effect.

Creating a development environment suited for reuse of code

As you have seen Oracle forms offers a number of ways to store your code, object libraries, plsql libraries forms etc but you should always create your forms in the same manner so others will be able to figure out what you have done. In figure18 I've drawn a "perfect" world example of how your environment should be configured.

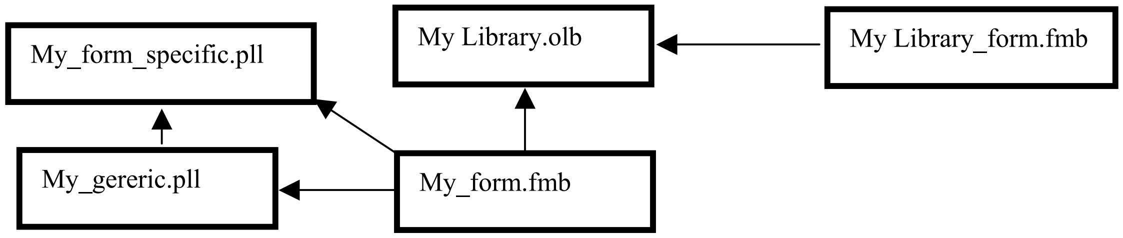


Figure 18 A perfect world.

You should separate your pl/sql into 2 plsql libraries, one for generic code and one for code specific for this form, if the specific pll-library are empty then you only need to attach the generic library. You also need to utilise your object library, and as mentioned earlier a library form might be handy to store your library objects.

Conclusion

In this article I've show how to write reusable code, demo'ed some features used in a real world life, and drawn a picture of an environment promoting the creation on reusable code. There are a lot of topics I would like to have covered in more detail, but time and space prevented me from doing so.

It is clear to me that j*developer over the next years will be more and more present in the custom development scene, but still I thing there is a place for the "classical" Oracle development tools. I'm sure that during the next couple of year j*developer environment will evolve to be more and more productive, so if the non-java people out there will continue to use the classical toolset, now is the time to gear (even more) up.

An appealing thought is it to create an intercommunity with the same values and thoughts that the Java people, and then everyone could benefit from this, I will think about it.

Rune is currently working as a senior developer at NNE A/S, a company specializing in delivering pharmaceutical plants in no time. Rune can be reached at runm@nne.dk.