This is part 2 of our series Workflow for eBS DBA's. The first part can be found here.

In this part, we are going to see how the process we built in the first part can be started. And how we find information about it in the wf_* tables.

In the previous part we built an itemtype and a process. You'll remember that the itemtype is the container that holds the other elements.

We can find the itemtype in our database with:

```
select wi.name
,      wi.persistence_type
,      wi.persistence_days
,      wit.display_name
,      wit.description
From   wf_item_types wi
,      wf_item_types_tl wit
where  wi.name=wit.name
and    wit.language='US'
and    wi.name='DBA_TYPE';
```

```
NAME     PERSISTE PERSISTENCE_DAYS DISPLAY_NAME    DESCRIPTION
-------- -------- ---------------- --------------- --------------------------------------------------
DBA_TYPE TEMP     0                DBA Itemtype    Itemtype to hold DBA processes and functions
```

So what can we find out about this item_type. We created one process within this item_type. Because processes can also be activities of another process, we need to identify the runnable processes. This is done by looking for the 'ROOT' processes:

```
select process_name
,      process_version
,      activity_item_type
,      activity_name
,      instance_id
,      instance_label
From   wf_process_activities
where  process_item_type='DBA_TYPE'
and    process_name='ROOT';
```

```
PROCESS_NAME       PROCESS_VERSION    ACTIVITY   ACTIVITY_NAME                 INSTANCE_ID   INSTANCE_LABEL
------------------ ------------------ ---------- ----------------------------- ------------- ----------------
ROOT                               1 DBA_TYPE   DBA_MAIN_PROCESS                    787224 DBA_MAIN_PROCESS
```

This shows us that there is one runnable process within this item_type, called: ‘DBA_MAIN_PROCESS’.
So how do we find out the definition of this process? First we need to find the right process_version.

```
select item_type
,      name
,      version
,      type
,      begin_date
,      end_date
From   wf_activities
where  name ='DBA_MAIN_PROCESS'
and    end_date is null;
```

```
ITEM_TYPE NAME                           VERSION              TYPE     BEGIN_DATE              END_DATE
--------- ------------------------------ -------------------- -------- ----------------------- -------------------------
DBA_TYPE  DBA_MAIN_PROCESS               1                    PROCESS  04-10-09
```

Note that I selected the version (which is 1 now). And I added ‘end_date is null’. When we update the process_definition, workflow will add new versions and end_date the previous one. This datamodel allows the workflow engine to run its own versioning.
Once a process is started (actually an item created), it will check for the latest version of the process. And it will keep that definition of the process. Even if halfway the runtime of the item, a new definition of the process is loaded to the database.
Now that we know the version (1), we can start looking for the starting point. This is where the start (and end) activities come into the play.

```
select process_name
,      process_version
```

```
,      activity_item_type
,      activity_name
,      instance_id
,      instance_label
From   wf_process_activities
where  process_item_type='DBA_TYPE'
and    process_name='DBA_MAIN_PROCESS'
and    process_version=1
and    start_end='START';
```

```
PROCESS_NAME                   PROCESS_VERSION ACTIVITY ACTIVITY_NAME                  INSTANCE_ID INSTANCE_LABEL
------------------------------ --------------- -------- ------------------------------ ----------- --------------
DBA_MAIN_PROCESS                             1 DBA_TYPE START                              787223 START
```

This selects the first activity for our version of the process. I also selected the instance_id. The instance_id is unique for the combination of item_type, process_name, process_version and activity_name. In other words it uniquely defines our activity in that specific item.

From this point we can follow the process through wf_activity_transitions:

```
select from_process_activity
,      to_process_activity
,      result_code
from   wf_activity_transitions wit
connect by from_process_activity=prior to_process_activity
start with from_process_activity= 787223;
```

```
FROM_PROCESS_ACTIVITY TO_PROCESS_ACTIVITY RESULT_CODE
--------------------- ------------------- ------------------------------
               787223              787227 *
               787227              787225 *
```

This table includes the process_activities connected to each other. The default result_code is ‘*’. Later on we will see different result_codes.

In our process, we see that the flow goes from process_activity instance_id 787223 to 787227 to 787225. In your system, the id’s will be different, of course.

Here is a query I often use to show the process, including the detail on the process_activities:

```
select l1
,      process_item_type
,      process_name
,      process_version version
,      activity_item_type
,      activity_name
,      instance_id
,      result_code
,      to_process_activity
from (select t2.*
     ,       case when l1=mx and l2=2 then to_process_activity
                  else from_process_activity
             end pa
      from (select t1.*
            ,      count(l1) over () mx
            from   (select level l1
                    ,       from_process_activity
                    ,       result_code
                    ,       to_process_activity
                    From    wf_activity_transitions wat
                    connect by prior wat.to_process_activity=wat.from_process_activity
                    start with wat.from_process_activity=244748
                    ) t1
            ) t2
            join (select level l2 from dual connect by level<3) dummy on (l2=1 or l1=mx)
    ) wf_proc
,     wf_process_activities wpa
where wf_proc.pa=wpa.instance_id;
```

```
L1  PROCESS_ITEM_TYPE PROCESS_NAME     VERSION ACTIVITY_ITEM_TYPE ACTIVITY_NAME  INSTANCE_ID  RESULT_CODE TO_PROCESS_ACTIVITY
--- ----------------- ---------------- ------- ------------------ -------------- ------------ ----------- -------------------
1   DBA_TYPE          DBA_MAIN_PROCESS 1       DBA_TYPE           START          787223       *           787227
2   DBA_TYPE          DBA_MAIN_PROCESS 1       DBA_TYPE           INIT           787227       *           787225
2   DBA_TYPE          DBA_MAIN_PROCESS 1       DBA_TYPE           END            787225       *           787225
```

Even though it is interesting to see the flow. We still can't see what is actually being done. That information is stored in the wf_activities:

```
select l1
,      process_item_type
,      process_name
,      process_version
,      wpa.instance_id
,      wf_proc.result_code
,      wa.item_type
,      wa.name
,      wa.type
,      wa.function
from  (select t2.*
       ,      case when l1=mx and l2=2 then to_process_activity
                   else from_process_activity
              end pa
       from (select t1.*
             ,      count(l1) over () mx
             from   (select level l1
                     ,      from_process_activity
                     ,      result_code
                     ,      to_process_activity
                     From   wf_activity_transitions wat
                     connect by prior wat.to_process_activity=wat.from_process_activity
                     start with wat.from_process_activity=244748
                     ) t1
             ) t2
       join (select level l2 from dual connect by level<3) dummy on (l2=1 or l1=mx)
            ) wf_proc
    ,    wf_process_activities wpa join wf_activities wa on
(wpa.activity_item_type=wa.item_type and wpa.activity_name=wa.name and wa.end_date is null)
where wf_proc.pa=wpa.instance_id;
```

```
L1 PROCESS_ITEM_TYPE PROCESS_NAME      PROCESS_VERSION INSTANCE_ID  RESULT_CODE ITEM_TYPE NAME   TYPE     FUNCTION
-- ----------------- ----------------- --------------- ------------ ----------- --------- ------ -------- ----------------
1  DBA_TYPE          DBA_MAIN_PROCESS  1               787223       *           DBA_TYPE  START  FUNCTION WF_STANDARD.NOOP
2  DBA_TYPE          DBA_MAIN_PROCESS  1               787227       *           DBA_TYPE  INIT   FUNCTION XXX_WF_DBA.init
2  DBA_TYPE          DBA_MAIN_PROCESS  1               787225       *           DBA_TYPE  END    FUNCTION WF_STANDARD.NOOP
```

I took a little shortcut here. Can you see which one? In the next article, I'll expand the query to accommodate more complex processes.

So far we have seen the definition of the process. There is one component left. Remember that we created an attribute in the previous part? This is stored in WF_ITEM_ATTRIBUTES(_TL):

```
select item_type,name,sequence,type, format
from wf_item_attributes
where item_type='DBA_TYPE';
```

```
ITEM_TYPE NAME                           SEQUENCE              TYPE     FORMAT
--------- ------------------------------ --------------------- -------- -----------
DBA_TYPE  INSTANCE_NAME                  0                     VARCHAR2 10
```

Now that we have seen the definition of the process. It's time to make it run. To run an item, we call the WF_ENGINE. WF_ENGINE is one of the core packages of workflow, and it contains many of the API's to control the items.

The easiest procedure is the 'launchprocess'. This creates an item, and starts executing it. Alternatively you can use the separate 'createprocess' and 'startprocess' procedures. For the moment, I use 'launchprocess'.

The parameters for the procedure are: itemtype, itemkey, process, userkey and owner.
Itemtype is the itemtype to which our process belongs.
Itemkey is a unique identifier for items within this itemtype. You can choose your own itemkey, but a meaningful one is advisable.
Process is the name of the process that we want to run. Later on, we'll see how we can let the workflow engine choose the process automatically.
Userkey is an optional extra key to the item.
Owner is an optional 'owner role'. We'll see more about this later again.

With those parameters we can start our first workflow item:

```
Begin
       Wf_engine.launchprocess(itemtype=>'DBA_TYPE'
                               ,itemkey=>'1'
                               ,process=>'DBA_MAIN_PROCESS');
End;
```

Note 1: Be aware that the workflow engine does not issue a commit. It is also not supported to commit inside procedures called by the workflow engine. We'll see the reasons for this later.

Note 2: The item_key is a varchar2 value. This is important to remember when querying the tables, to avoid implicit type-conversions.

Let's see what has happened on the database level. First an item was created:

```
select item_type
,      item_key
,      root_activity
,      root_activity_version
,      begin_date
,      end_date
,      user_key
from   wf_items
where  item_type='DBA_TYPE';
```

```
ITEM_TYP ITEM_KEY   ROOT_ACTIVITY                 ROOT_ACTIVITY_VERSION BEGIN_DA END_DATE USER_KEY
-------- ---------- ---------------------------- --------------------- -------- -------- ---------
DBA_TYPE 1          DBA_MAIN_PROCESS                                 1 20-07-09 20-07-09
```

So we see that our item has been started and finished. (There are timestamps on the begin- and end-date).

To see what was actually executed, we need to look at wf_item_activity_statuses:

```
select item_type
,      item_key
,      process_activity
,      activity_status
,      activity_result_code
,      begin_date
,      end_date
,      execution_time
from wf_item_activity_statuses
where item_type='DBA_TYPE'
and item_key='1'
order by begin_date,execution_time;
```

```
ITEM_TYPE ITEM_KEY PROCESS_ACTIVITY ACTIVITY_STATUS ACTIVITY_RESULT_CODE BEGIN_DATE END_DATE  EXECUTION_TIME
--------- -------- ---------------- --------------- -------------------- ---------- --------- ----------------------
DBA_TYPE  1        787224           COMPLETE        #NULL                04-10-09   04-10-09  1
DBA_TYPE  1        787223           COMPLETE        #NULL                04-10-09   04-10-09  2
DBA_TYPE  1        787227           COMPLETE        COMPLETE             04-10-09   04-10-09  4
DBA_TYPE  1        787225           COMPLETE        #NULL                04-10-09   04-10-09  6
```

Here we see the activities that were executed. We see 4 rows. This is because the process itself is also returned as an item_activity.

Some things to note: The process_activity refers to the instance_id on wf_process_activities. This is part of the workflow versioning.

The execution_time is set on creation and update of the item_activity_status. It is a 'global' counter for the number of times an item_activity_status is updated during the runtime of the workflow engine. So an update of an item_activity_status for any workflow item will update the counter.
That way you will always get a correct sequence from wf_item_activity_statuses by ordering on begin_date,execution_time.

A little bit more complete query to see the progress of the item, including the activities:

```
select wias.item_type
,      wias.item_key
,      wat.display_name
,      case when wpa.start_end is not null then wpa.start_end else wa.function end function
```

```
,      wias.begin_date
,      wias.end_date
,      wias.activity_status
,      wias.activity_result_code
from wf_item_activity_statuses wias
join wf_items wi on (wias.item_type=wi.item_type and wias.item_key=wi.item_key)
join wf_process_activities wpa on (wias.process_activity=wpa.instance_id)
join wf_activities wa on (wpa.activity_item_type=wa.item_type
                      and wpa.activity_name=wa.name
                  and wi.begin_date between wa.begin_date and nvl(wa.end_date,wi.begin_date+1))
join wf_activities_tl wat on (wa.item_type=wat.item_type
                          and wa.name=wat.name
                          and wa.version=wat.version
                          and wat.language='US')
where wias.item_type='DBA_TYPE'
and   wias.item_key='1'
order by wias.begin_date,wias.execution_time;

ITEM_TYPE ITEM_KEY FUNCTION        BEGIN_DATE END_DATE ACTIVITY_STATUS ACTIVITY_RESULT_CODE
--------- -------- --------------- ---------- -------- --------------- ------------------------------
DBA_TYPE  1                       04-10-09   04-10-09 COMPLETE        #NULL
DBA_TYPE  1        START          04-10-09   04-10-09 COMPLETE        #NULL
DBA_TYPE  1        XXX_WF_DBA.init 04-10-09   04-10-09 COMPLETE        COMPLETE
DBA_TYPE  1        END            04-10-09   04-10-09 COMPLETE        #NULL
```

At the end of this article a few things remain.
We still need to see the versioning in action. And it's time to look at our item attributes.

To see the versioning, I added an extra procedure to our package: init2. This procedure will not set the attribute value to the instance name, but to the instance host_name:

```
PROCEDURE init2(
    p_item_type IN VARCHAR2 ,
    p_item_key  IN VARCHAR2 ,
    p_actid     IN NUMBER ,
    p_funcmode  IN VARCHAR2 ,
    p_result OUT VARCHAR2)
IS
BEGIN
  IF p_funcmode='RUN' THEN
       wf_engine.SetItemAttrText(itemtype=>p_item_type
                                ,itemkey =>p_item_key
                                ,aname=>'INSTANCE_NAME'
                                ,avalue=> sys_context('USERENV','SERVER_HOST'));
  end if;
  p_result:=wf_engine.eng_completed;
END;
```

Now we create the first item without actually launching it. This item will use the original 'init' procedure.

```
Begin
Wf_engine.createprocess(itemtype=>'DBA_TYPE', itemkey=>'2',process=>'DBA_MAIN_PROCESS');
End;
Commit;
```

Now we return to the workflow builder. Here we update the 'Initialize workflow' function to run XXX_WF_DBA.init2.

The first thing is to check our version:

```
select item_type
,      name
,      version
,      type
,      begin_date
,      end_date
From   wf_activities
where  name ='DBA_MAIN_PROCESS'
and    end_date is null;

ITEM_TYPE NAME                           VERSION              TYPE     BEGIN_DATE              END_DATE
--------- ------------------------------ -------------------- -------- ----------------------- -------------------------
DBA_TYPE  DBA_MAIN_PROCESS               2                    PROCESS  04-10-09
```

The process is now version 2. Version 1 has been end-dated at the time version 2 began. When we check the definitions (query above), we see that now init2 is called:

```
PROCESS_ITEM_TYPE PROCESS_NAME      PROCESS_VERSION INSTANCE_ID RESULT_CODE ITEM_TYPE NAME   TYPE     FUNCTION
----------------- ----------------- --------------- ----------- ----------- --------- ------ -------- -----------------
DBA_TYPE          DBA_MAIN_PROCESS  2               787235      *           DBA_TYPE  START  FUNCTION WF_STANDARD.NOOP
DBA_TYPE          DBA_MAIN_PROCESS  2               787233      *           DBA_TYPE  INIT   FUNCTION XXX_WF_DBA.init2
DBA_TYPE          DBA_MAIN_PROCESS  2               787231      *           DBA_TYPE  END    FUNCTION WF_STANDARD.NOOP
```

Let's create a new item, and see the difference between them:

```
Begin
Wf_engine.createprocess(itemtype=>'DBA_TYPE', itemkey=>'3',process=>'DBA_MAIN_PROCESS');
End;

begin
wf_engine.startprocess(itemtype=>'DBA_TYPE',itemkey=>'2');
end;
```

And of course repeat the startprocess for item_key '3'.

Let's look at the progress for both of them:

```
select wias.item_type
,      wias.item_key
,      wat.display_name
,      case when wpa.start_end is not null then wpa.start_end else wa.function end function
,      wias.begin_date
,      wias.end_date
,      wias.activity_status
,      wias.activity_result_code
from wf_item_activity_statuses wias
join wf_items wi on (wias.item_type=wi.item_type and wias.item_key=wi.item_key)
join wf_process_activities wpa on (wias.process_activity=wpa.instance_id)
join wf_activities wa on (wpa.activity_item_type=wa.item_type
                      and wpa.activity_name=wa.name
                  and wi.begin_date between wa.begin_date and nvl(wa.end_date,wi.begin_date+1))
join wf_activities_tl wat on (wa.item_type=wat.item_type
                          and wa.name=wat.name
                          and wa.version=wat.version
                          and wat.language='US')
where wias.item_type='DBA_TYPE'
and   wias.item_key in ('2', '3')
order by wias.begin_date,wias.execution_time;
```

```
ITEM_TYPE ITEM_KEY DISPLAY_NAME         FUNCTION          BEGIN_DATE END_DATE ACTIVITY_STATUS ACTIVITY_RESULT_CODE
--------- -------- -------------------- ----------------- ---------- -------- --------------- ------------------------------
DBA_TYPE  2        DBA Main Process                       04-10-09   04-10-09 COMPLETE        #NULL
DBA_TYPE  2        Start                START             04-10-09   04-10-09 COMPLETE        #NULL
DBA_TYPE  2        Initialize workflow  XXX_WF_DBA.init   04-10-09   04-10-09 COMPLETE        COMPLETE
DBA_TYPE  2        End                  END               04-10-09   04-10-09 COMPLETE        #NULL
DBA_TYPE  3        DBA Main Process                       04-10-09   04-10-09 COMPLETE        #NULL
DBA_TYPE  3        Start                START             04-10-09   04-10-09 COMPLETE        #NULL
DBA_TYPE  3        Initialize workflow  XXX_WF_DBA.init2  04-10-09   04-10-09 COMPLETE        COMPLETE
DBA_TYPE  3        End                  END               04-10-09   04-10-09                 COMPLETE        #NULL
```

As expected, item_key 2 has executed the original version with XXX_WF_DBA.init. And item_key 3 has executed the newer XXX_WF_DBA.init2.

Both of them should have assigned values to the item attribute 'DB Instance name'. So let's take a look.

We already saw that item attributes are stored in WF_ITEM_ATTRIBUTES. Their values are stored in WF_ITEM_ATTRIBUTE_VALUES:

```
select item_type,item_key,name,text_value
from wf_item_attribute_values
where item_type='DBA_TYPE'
order by item_key;
```

```
ITEM_TYPE ITEM_KEY NAME                           TEXT_VALUE
--------- -------- ------------------------------ ----------------------------------------
DBA_TYPE  2        #SCHEMA                        APPS
DBA_TYPE  2        .ADMIN_KEY                     339029335872007084374783033858530570 47
DBA_TYPE  2        .MONITOR_KEY                   195019319494724212237380720564043747236
DBA_TYPE  2        INSTANCE_NAME                  VIS
DBA_TYPE  3        #SCHEMA                        APPS
DBA_TYPE  3        .ADMIN_KEY                     305730102624958153831937205934587328045
DBA_TYPE  3        .MONITOR_KEY                   241877462733314091534969698772096556724
DBA_TYPE  3        INSTANCE_NAME                  server-1
```

There are some attribute values that are not listed in WF_ITEM_ATTRIBUTES. These are seeded attributes and they are used in the workflow status monitoring screens.
For us, also the value of INSTANCE_NAME is shown. And indeed, item_key 2 shows ‘VIS’, which is the instance_name. And item_key 3 shows ‘server-1’, which is the DB-server for that instance.

With that we end part 2 of this series.
In the next part we are going to look at the workflow constructs.