

Workflow for ebs DBA's (part 3)

In this part of our series (previous parts to be found [here](#) and [here](#)) we are going to look into what I call workflow constructs. These are the constructs that determine the flow of your process. They include if (or case) statements, and/or gates, loops and finally waits and holds.

Let's start with the

if / Case

We pick up our initial workflow from [part 2](#). In the original function, we assigned the DB instance name to an item attribute. Now we will build a 'compare' function that tests for the DB name and processes only if the correct name is found.

The workflow engine delivers standard functions to compare values to each other. These are called by WF_ENGINE.compare. There are some other compare functions, but we won't discuss these yet. Many of the functions in the WF_ENGINE are available from the 'Standard' itemtype. This itemtype is already available in eBS databases. Alternatively, it can be loaded from the workflow file \$ORACLE_HOME\wf\data\US\WFSTD.wft.

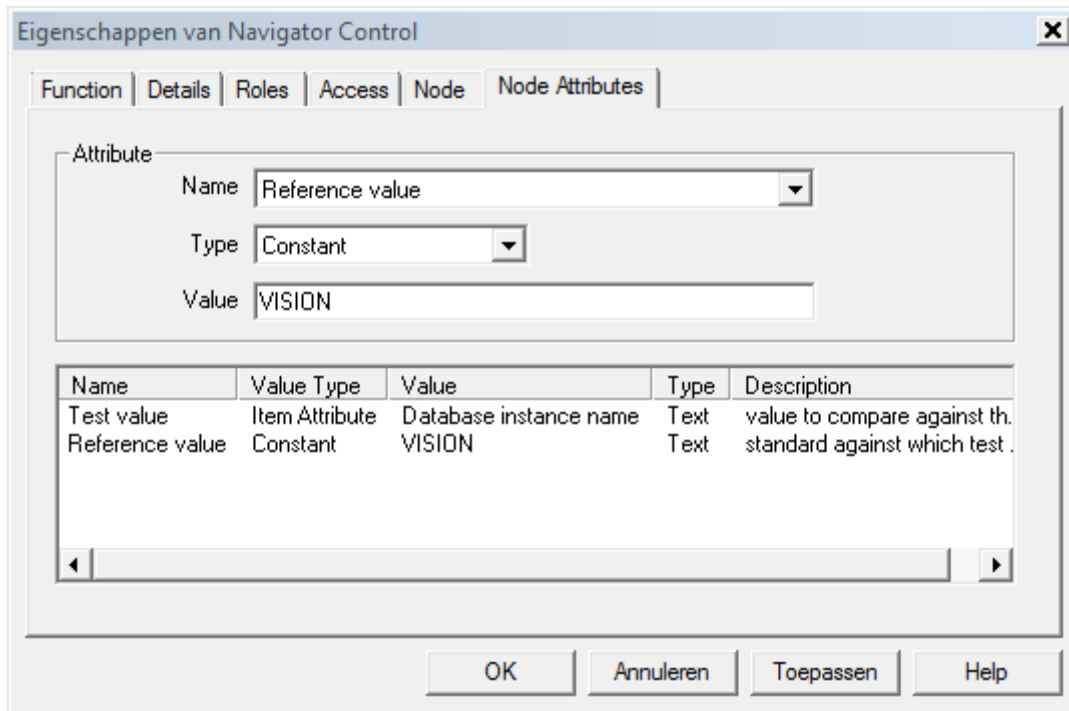
So we add an extra function to our process:

The screenshot shows a dialog box titled "Eigenschappen van Navigator Control" with a close button (X) in the top right corner. The dialog has several tabs: "Function", "Details", "Roles", "Access", "Node", and "Node Attributes". The "Function" tab is selected. The fields and their values are as follows:

| Field | Value | Action |
|---------------|--------------------------|--------|
| Item Type | Standard | Edit |
| Internal Name | COMPARETEXT | New |
| Display Name | Compare Text | |
| Description | Compare two text values. | |
| Icon | COMPARE.ICO | Browse |
| Function Name | WF_STANDARD.COMPARE | |
| Function Type | PL/SQL | |
| Result Type | Comparison | Edit |
| Cost | 0.01 | |

At the bottom of the dialog, there are four buttons: "OK", "Annuleren", "Toepassen", and "Help".

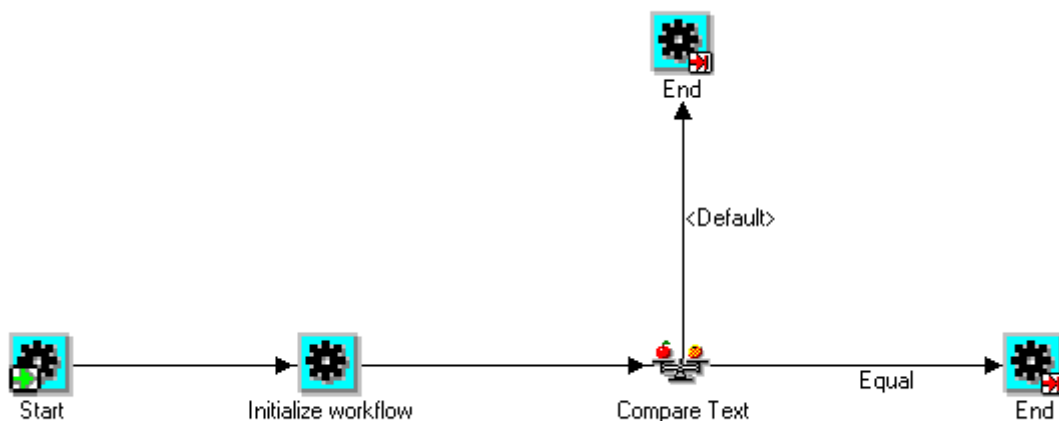
We set the input parameters on the Node Attributes tab. We need a Test- and Reference value. Either can be either a constant or an item attribute:



The result type for the Compare is 'Comparison'. Which has lookup values 'Equal', 'Greater than', 'Less than' or 'Null'. Of course there is also the default result '*'.

In our case we need the Equal result to continue the workflow. Since there is no result value for 'Not Equal' we use the default for the other exit.

The result looks like this:



Instead of the <Default> result value, we could have created 2 exits with GT and LT. But in the workflow engine exact matches take precedence over the <Default>.

Before we run the process, let's first look at the definition in the WF_* tables. First we can see the transitions from the compare function in WF_ACTIVITY_TRANSITIONS.

In [part 2](#) of the series we saw the query to see the definition of a process. However, this query didn't take into account the possibility of more than one exit on an activity. I adjusted the query a bit for this:

```

select wpa3.process_item_type
,      wpa3.process_name
,      wpa3.process_version
,      wpa3.instance_id
,      wa3.item_type
,      wa3.type
,      wa3.name
,      wpa3.instance_label
,      wa3.function
,      trans.result_code
,      trans.p_from_act
,      trans.p_res_code from_res_code
from   (
select r*100+l2 cntr
,      wat2.result_code
,      wat2.p_res_code
,      wat2.p_from_act
,      case when wat2.start_end = 'END' and d.l2=2
             then wat2.to_process_activity
             else wat2.from_process_activity
             end e_proc
from   (select wat1.from_process_activity
,      wat1.to_process_activity
,      wat1.result_code
,      wpa2.start_end
,      rownum r
,      prior wat1.result_code p_res_code
,      prior wat1.from_process_activity p_from_act
from   wf_activity_transitions wat1
join   wf_process_activities wpa2
on     (wat1.to_process_activity=wpa2.instance_id)
connect by prior wat1.to_process_activity = wat1.from_process_activity
start with wat1.from_process_activity=(select wpa1.instance_id
                                       from wf_process_activities wpa1
                                       join wf_activities wal
                                       on (wpa1.process_item_type=wal.item_type
                                           and wpa1.process_name=wal.name
                                           and wpa1.process_version=wal.version
                                           and wal.end_date is null)
                                       where wpa1.process_item_type='DBA_TYPE'
                                       and wpa1.process_name='DBA_MAIN_PROCESS'
                                       and wpa1.start_end='START')
                                       ) wat2

join   (select level l2
from   dual
connect by level<3) d
on     ((start_end='END' and d.l2=2) or d.l2=1)
) trans
join   wf_process_activities wpa3
on     (trans.e_proc = wpa3.instance_id)
join   wf_activities wa3
on     (wpa3.activity_item_type=wa3.item_type
and    wpa3.activity_name=wa3.name
and    wa3.end_date is null)

order by cntr;

```

| ITEM_TYPE | PROCESS_NAME | VE | INSTAN | TYPE | NAME | INSTANCE_LABEL | FUNCTION | RESULT_C | P_FROM_A | P_RES_C |
|-----------|------------------|----|--------|----------|-------------|----------------|---------------------|----------|----------|---------|
| DBA_TYPE | DBA_MAIN_PROCESS | 7 | 787481 | FUNCTION | START | START | WF_STANDARD.NOOP | * | (null) | (null) |
| DBA_TYPE | DBA_MAIN_PROCESS | 7 | 787479 | FUNCTION | INIT | INIT | XXX_WF_DBA.init | * | 787481 | * |
| DBA_TYPE | DBA_MAIN_PROCESS | 7 | 787483 | FUNCTION | COMPARETEXT | COMPARETEXT | WF_STANDARD.COMPARE | * | 787479 | * |
| DBA_TYPE | DBA_MAIN_PROCESS | 7 | 787485 | FUNCTION | END | END | WF_STANDARD.NOOP | * | 787479 | * |
| DBA_TYPE | DBA_MAIN_PROCESS | 7 | 787483 | FUNCTION | COMPARETEXT | COMPARETEXT | WF_STANDARD.COMPARE | EQ | 787479 | * |
| DBA_TYPE | DBA_MAIN_PROCESS | 7 | 787487 | FUNCTION | END | END-1 | WF_STANDARD.NOOP | EQ | 787479 | * |

But where do we find the information about comparetext. First, we entered some node attributes, for the values to compare. The node attributes are visible in the workflow builder as attributes, attached to the function. They are not stored in wf_item_attributes. But in wf_activity_attributes. For our comparetext, we can see them with:

```
select activity_item_type
,      activity_name
,      activity_version
,      name
,      sequence
,      type
,      value_type
from wf_activity_attributes
where (activity_item_type,activity_name,activity_version)
in (select wa.item_type,wa.name,wa.version
    from wf_activities wa
    join wf_activities wa2
on (wa.begin_date<=wa2.begin_date and (wa.end_date>wa2.begin_date or wa.end_date is null))
    join wf_process_activities wpa
on (wpa.activity_item_type=wa.item_type and wpa.activity_name=wa.name)
    where wa2.item_type=wpa.process_item_type
    and wa2.name=wpa.process_name
    and wa2.version=wpa.process_version
    and wpa.process_item_type='DBA_TYPE'
    and wpa.process_name='DBA_MAIN_PROCESS'
    and wpa.activity_name='COMPARETEXT'
    and wpa.process_version=7);
```

| ACTIVITY_ITEM_TYPE | ACTIVITY_NAME | ACTIVITY_VERSION | NAME | SEQUENCE | TYPE | VALUE_TYPE |
|--------------------|---------------|------------------|--------|----------|----------|------------|
| WFSTD | COMPARETEXT | 335 | VALUE1 | 0 | VARCHAR2 | CONSTANT |
| WFSTD | COMPARETEXT | 335 | VALUE2 | 1 | VARCHAR2 | CONSTANT |

The sequence is used for ordering the attributes, both in the workflow builder and the compare function. The values for these activity_attributes are stored in WF_ACTIVITY_ATTR_VALUES. The key for this table is the process_activity_id, which we have seen above:

```
select process_activity_id,name,value_type,text_value
from wf_activity_attr_values
where process_activity_id=787483;
```

| PROCESS_ACTIVITY_ID | NAME | VALUE_TYPE | TEXT_VALUE |
|---------------------|--------|------------|---------------|
| 787483 | VALUE1 | ITEMATTR | INSTANCE_NAME |
| 787483 | VALUE2 | CONSTANT | VIS |

The instance_name is based on an item_attribute. Therefore it is not yet known at this stage. It will be set during runtime.

Let's see what happens when we run the process now:

```
Begin
    Wf_engine.launchprocess(itemtype=>'DBA_TYPE',itemkey=>'4',process=>'DBA_MAIN_TYPE');
End;

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='4'
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 | 4 | DBA Main Process | | 11-10-09 | 11-10-09 | COMPLETE | #NULL |
| DBA_TYPE | 4 | Start | START | 11-10-09 | 11-10-09 | COMPLETE | #NULL |
| DBA_TYPE | 4 | Initialize workflow | XXX_WF_DBA.init | 11-10-09 | 11-10-09 | COMPLETE | COMPLETE |
| DBA_TYPE | 4 | Compare Text | WF_STANDARD.COMPARE | 11-10-09 | 11-10-09 | COMPLETE | EQ |
| DBA_TYPE | 4 | End | END | 11-10-09 | 11-10-09 | COMPLETE | #NULL |

After the compare function that functions as an if/case statement, we will see the logical gateways for and / or.

And / Or

These functions are based on joining two different tracks from a workflow. In the previous section we created an exit for EQ and one for '*', default. Both exits led to a different end-node. Now we will split a process and let both tracks lead to the same end. The join can either be based on an 'And' or an 'Or' function.

First we need to set up some extra functions. We will be counting the number of invalid objects in the database and store the number in an item attribute. After that, we return to the main track again. To follow the progress of the flow we create an extra tracking function.

First a tracking table:

```

create table xxx_track_flow (id number
        ,item_type varchar2(8)
        ,item_key varchar2(240)
        ,activity_name varchar2(30)
        ,activity_version number
        ,process_name varchar2(30)
        ,instance_id number
        ,instance_label varchar2(30)
        ,funcmode varchar2(30)
        );

create sequence xxx_track_flow_s start with 1 increment by 1;

```

And the procedure itself. I added this to the XXX_WF_DBA package that we created in the second part of this series.

```

Create Procedure track_flow_progress (p_item_type IN VARCHAR2
        ,p_item_key IN VARCHAR2
        ,p_actid IN NUMBER
        ,p_funcmode IN VARCHAR2
        ,p_result OUT VARCHAR2) IS
v_activity_name varchar2(30);
v_activity_version number;
v_process_name varchar2(30);
v_instance_label varchar2(30);
begin
    select activity_name,wa.version,process_name,instance_label
    into v_activity_name, v_activity_version, v_process_name, v_instance_label
    from wf_process_activities wpa join wf_activities wa
    on (wpa.activity_item_type=wa.item_type
        and wpa.activity_name=wa.name
        and wpa.process_version=wa.version)
    where wpa.instance_id=p_actid;

```

```

insert into xxx_track_flow (id, item_type, item_key, activity_name, activity_version,
                           process_name, instance_id, instance_label, funcmode)
values (xxx_track_flow_s.nextval, p_item_type, p_item_key, v_activity_name,
       v_activity_version, v_process_name, p_actid, v_instance_label, p_funcmode);
end;

```

This will record the basic information about the process activity being called. We put this in a new function in our process:

The screenshot shows a 'Navigator Control Properties' dialog box with the following fields and values:

- Internal Name: TRACK_FLOW
- Display Name: Track flow progress
- Description: Track flow progress
- Icon: FUNCTION.ICO (with a gear icon and a 'Browse' button)
- Function Name: XXX_WF_DBA.track_flow_progress
- Function Type: PL/SQL
- Result Type: <None> (with an 'Edit' button)
- Cost: 0.00

At the bottom, there are buttons for OK, Cancel, Apply, and Help.

Now we create a function to check on the number of invalid objects. Also part of the XXX_WF_DBA package.

```

PROCEDURE CHECK_INVALIDS(p_item_type IN VARCHAR2
                        ,p_item_key IN VARCHAR2
                        ,p_actid IN NUMBER
                        ,p_funcmode IN VARCHAR2
                        ,p_result OUT VARCHAR2) IS
v_invalids varchar2(1) := 'N';
BEGIN
  IF p_funcmode='RUN' then
    select case when count(*)>0 then 'Y' else 'N' end inv
    into v_invalids
    from all_objects
    where status='INVALID';
  END IF;
  p_result := v_invalids;
END;
/

```

Of course we also need to create a result type. In this case, it will contain a 'Y' or 'N':

The screenshot shows a dialog box titled "Eigenschappen van Navigator Control" with a close button (X) in the top right corner. The "Lookup Type" tab is selected, and the "Access" sub-tab is also visible. The dialog contains three input fields: "Internal Name" with the value "YES_NO", "Display Name" with the value "Yes No", and "Description" with the value "Yes or no". At the bottom, there are four buttons: "OK", "Annuleren", "Toepassen", and "Help".

Rightclick the lookup type to add the lookup codes for Y and N:

The screenshot shows the same dialog box, but now the "Lookup Code" tab is selected. The "Lookup Type" field contains "Yes No", "Internal Name" contains "Y", "Display Name" contains "Yes", and "Description" contains "Yes". The "Access" sub-tab is no longer visible. The "OK", "Annuleren", "Toepassen", and "Help" buttons remain at the bottom.

Finally add CHECK_INVALIDS:

Navigator Control Properties

Activity | Details | Roles | Access

Internal Name: XXX_CHECK_INVALIDS

Display Name: Check Invalid Objects

Description: Check for Invalid Objects

Icon: FUNCTION.ICO [Browse]

Function Name: XXX_WF_DBA.check_invalids

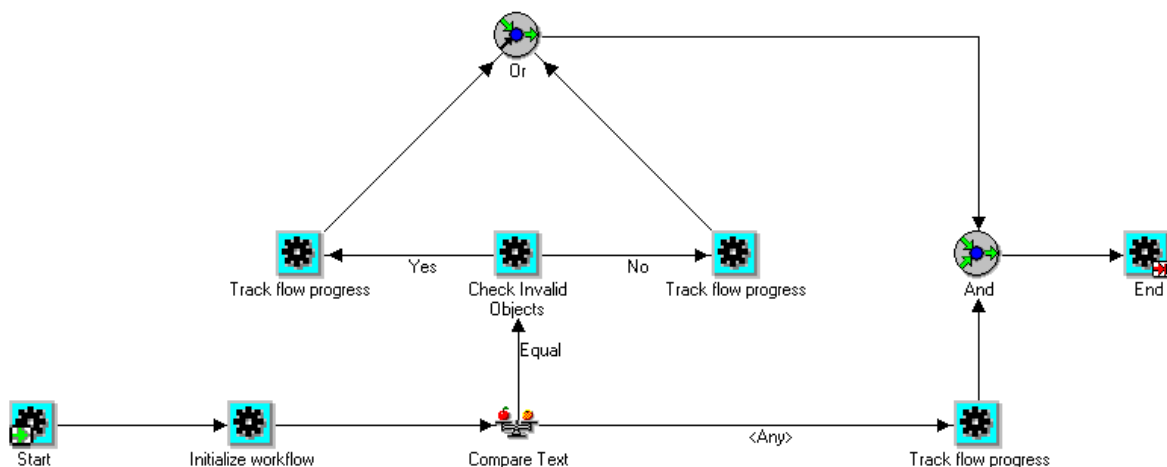
Function Type: PL/SQL

Result Type: Yes No [Edit]

Cost: 0.00

[OK] [Cancel] [Apply] [Help]

Now we are ready to expand our process:



Quite a lot has changed here. Let's start with the 'triangle'. After validating the database name, we check for the invalid objects. This has 2 possible exits. There are either invalid objects or not. Both exits lead to an 'OR' function via the 'Track flow progress'. The OR function will continue when either of the 'in'-transitions is activated.

On the compare, we changed the '*' exit to an 'Any' exit. This exit will always be called. After another 'Track flow progress' we reach an 'And' activity. This function will only continue when ALL incoming transitions are activated.

In our case, the item will complete successfully. And we will see two Track flow records. Let's first see the definition in the WF_* tables:

| VERSION | INSTANCE | ITEM_TYPE | TYPE | INSTANCE_LABEL | FUNCTION | RESULT | FROM_ACT | FROM_RES_CODE |
|---------|----------|-----------|----------|--------------------|-------------------------------|--------|----------|---------------|
| 11 | 787731 | DBA_TYPE | FUNCTION | START | WF_STANDARD.NOOP | * | | |
| 11 | 787729 | DBA_TYPE | FUNCTION | INIT | XXX_WF_DBA.init | * | 787731 | * |
| 11 | 787727 | WFSTD | FUNCTION | COMPARETEXT | WF_STANDARD.COMPARE | EQ | 787729 | * |
| 11 | 787733 | DBA_TYPE | FUNCTION | XXX_CHECK_INVALIDS | XXX_WF_DBA.check_invalids | N | 787727 | EQ |
| 11 | 787737 | DBA_TYPE | FUNCTION | TRACK_FLOW-1 | XXX_WF_DBA.track_flow_progres | * | 787733 | N |
| 11 | 787739 | WFSTD | FUNCTION | OR | WF_STANDARD.ORJOIN | * | 787737 | * |
| 11 | 787741 | WFSTD | FUNCTION | AND | WF_STANDARD.ANDJOIN | * | 787739 | * |
| 11 | 787743 | DBA_TYPE | FUNCTION | END | WF_STANDARD.NOOP | * | 787739 | * |
| 11 | 787733 | DBA_TYPE | FUNCTION | XXX_CHECK_INVALIDS | XXX_WF_DBA.check_invalids | Y | 787727 | EQ |
| 11 | 787735 | DBA_TYPE | FUNCTION | TRACK_FLOW | XXX_WF_DBA.track_flow_progres | * | 787733 | Y |
| 11 | 787739 | WFSTD | FUNCTION | OR | WF_STANDARD.ORJOIN | * | 787735 | * |
| 11 | 787741 | WFSTD | FUNCTION | AND | WF_STANDARD.ANDJOIN | * | 787739 | * |
| 11 | 787743 | DBA_TYPE | FUNCTION | END | WF_STANDARD.NOOP | * | 787739 | * |
| 11 | 787727 | WFSTD | FUNCTION | COMPARETEXT | WF_STANDARD.COMPARE | #ANY | 787729 | * |
| 11 | 787745 | DBA_TYPE | FUNCTION | TRACK_FLOW-2 | XXX_WF_DBA.track_flow_progres | * | 787727 | #ANY |
| 11 | 787741 | WFSTD | FUNCTION | AND | WF_STANDARD.ANDJOIN | * | 787745 | * |
| 11 | 787743 | DBA_TYPE | FUNCTION | END | WF_STANDARD.NOOP | * | 787745 | * |

From here we can see that either TRACK_FLOW-1 (787737) or TRACK_FLOW (787735) will be called after the 'XXX_CHECK_INVALIDS'. TRACK_FLOW-2 (787745) will be called from the 'Any'-exit on the COMPARETEXT.

Let's see what happens when we run the process now.

| ITEM_KEY | DISPLAY_NAME | INSTANCE | INSTANCE_LABEL | BEGIN_DATE | END_DATE | STATUS | RESULT |
|----------|-----------------------|----------|--------------------|---------------------|---------------------|----------|----------|
| 6 | DBA Main Process | 787224 | DBA_MAIN_PROCESS | 11-10-2009 19:48:15 | 11-10-2009 19:48:15 | COMPLETE | #NULL |
| 6 | Start | 787731 | START | 11-10-2009 19:48:15 | 11-10-2009 19:48:15 | COMPLETE | #NULL |
| 6 | Initialize workflow | 787729 | INIT | 11-10-2009 19:48:15 | 11-10-2009 19:48:15 | COMPLETE | COMPLETE |
| 6 | Compare Text | 787727 | COMPARETEXT | 11-10-2009 19:48:15 | 11-10-2009 19:48:15 | COMPLETE | EQ |
| 6 | Track flow progress | 787745 | TRACK_FLOW-2 | 11-10-2009 19:48:15 | 11-10-2009 19:48:15 | COMPLETE | |
| 6 | Check Invalid Objects | 787733 | XXX_CHECK_INVALIDS | 11-10-2009 19:48:15 | 11-10-2009 19:48:15 | COMPLETE | Y |
| 6 | Track flow progress | 787735 | TRACK_FLOW | 11-10-2009 19:48:15 | 11-10-2009 19:48:15 | COMPLETE | |
| 6 | Or | 787739 | OR | 11-10-2009 19:48:15 | 11-10-2009 19:48:15 | COMPLETE | #NULL |
| 6 | And | 787741 | AND | 11-10-2009 19:48:15 | 11-10-2009 19:48:15 | COMPLETE | #NULL |
| 6 | End | 787743 | END | 11-10-2009 19:48:15 | 11-10-2009 19:48:15 | COMPLETE | #NULL |

Some interesting things here. We see that the COMPARETEXT first executed the 'Any' exit. When it reached the 'AND' function, the workflow engine started executing the 'EQ' exit. This led to the Check-invalids, which went through the 'Y' exit. When it reached the 'OR' function, it continued to the 'AND' again, which finally led to the 'END' function.

Note that the WF_ITEM_ACTIVITY_STATUSES only registers the last exit that it took. The 'Any' exit is not registered, only the functions executed afterwards.

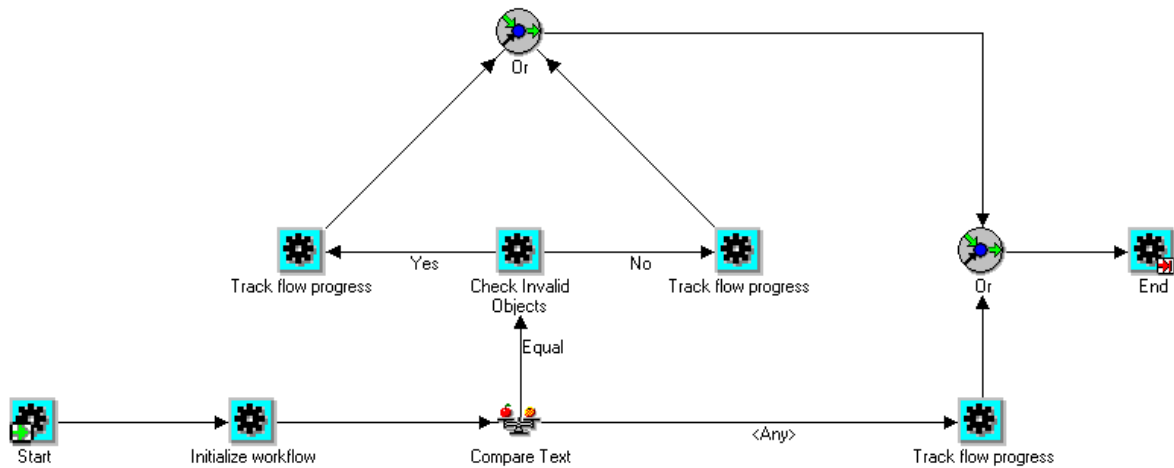
We can also verify that the above is indeed what happened, by checking the 'XXX_TRACK_FLOW' table:

```
select *
from xxx_track_flow
where item_type='DBA_TYPE'
and item_key='6';
```

| ID | ITEM_TYPE | ITEM_KEY | ACTIVITY_NAME | VERSION | PROCESS_NAME | INSTANCE | INSTANCE_LABEL | FUNCMODE |
|----|-----------|----------|---------------|---------|------------------|----------|----------------|----------|
| 6 | DBA_TYPE | 6 | TRACK_FLOW | 4 | DBA_MAIN_PROCESS | 787745 | TRACK_FLOW-2 | RUN |
| 7 | DBA_TYPE | 6 | TRACK_FLOW | 4 | DBA_MAIN_PROCESS | 787735 | TRACK_FLOW | RUN |

In case of multiple transitions from one activity, the workflow engine will start processing them in turns. When a transition leads to a point where the workflow engine can't continue, for example an 'End' or 'And' activity, it will process the next transition. In the next part of the series we will also see other activities like 'Defer' and 'Block' that have the same effect.

To emphasize the consequences of this, we will replace the 'And' activity with an 'Or':



Now when we run the process, we see the following results:

| ITEM_KEY | DISPLAY_NAME | FUNCTION | BEGIN_DATE | END_DATE | STATUS | ACTIVITY_RESULT_CODE |
|----------|-----------------------|-----------------------------|------------|----------|----------|----------------------|
| 6 | DBA Main Process | | 12-10-09 | 12-10-09 | COMPLETE | #NULL |
| 6 | Start | START | 12-10-09 | 12-10-09 | COMPLETE | #NULL |
| 6 | Initialize workflow | XXX_WF_DBA.init | 12-10-09 | 12-10-09 | COMPLETE | COMPLETE |
| 6 | Compare Text | WF_STANDARD.COMPARE" | 12-10-09 | 12-10-09 | COMPLETE | EQ |
| 6 | Track flow progress | XXX_WF_DBA.track_flow_progr | 12-10-09 | 12-10-09 | COMPLETE | |
| 6 | Or | WF_STANDARD.ORJOIN | 12-10-09 | 12-10-09 | COMPLETE | #NULL |
| 6 | End | END | 12-10-09 | 12-10-09 | COMPLETE | #NULL |
| 6 | Check Invalid Objects | XXX_WF_DBA.check_invalids | 12-10-09 | 12-10-09 | COMPLETE | #FORCE |

As you can see, the activity statuses have a logical inconsistency this time. Even though the result_code for the Compare is 'EQ', the only activities that were executed are from the 'Any' exit. When the workflow engine reached an 'END' node, it completed the whole item. The XXX_CHECK_INVALIDS was entered in the WF_ITEM_ACTIVITY_STATUSES. But it never actually executed. Instead, the workflow engine 'forced' its completion. As soon as the workflow engine reaches an 'End' node, it will force completion of all open activities.

Now that we have seen how the 'And' / 'Or' functions work, and some of their pitfalls, we can look into the last construct. That is the

Loop

Loops are among the most misused features in the workflow engine. This is because the loops are not only used for repeating functions. It is also used to re-do functions.

For this article, we'll expand our process to compile all invalid objects (assuming you have some, which will usually be the case for an eBS database) inside a loop.

First we need to set up some things again. We need a counter for the number of invalid objects. We'll store this in an item attribute:

The screenshot shows a dialog box titled "Navigator Control Properties" with a close button (X) in the top right corner. It has two tabs: "Attribute" and "Access", with "Access" selected. The "Access" tab contains the following fields:

- Item Type: DBA Item type
- Internal Name: INVALID_CNTR
- Display Name: Invalids counter
- Description: Invalids counter
- Type: Number (dropdown menu)
- Format: (empty text field)
- Default section:
 - Type: Constant (dropdown menu)
 - Value: (empty text field)

At the bottom of the dialog are four buttons: OK, Cancel, Apply, and Help.

Then we need a way to store the names of the invalid objects. Since the workflow item will be completely processed in one session, I will store them in a PL/SQL table. In the next part, we will see how to transfer control of an item to another session. In that case, you'll need an interim table, or you need to make use of the more advanced attribute types.

We first add a PL/SQL table to our XXX_WF_DBA package.

```
create or replace
PACKAGE XXX_WF_DBA AS

Type R_invalids Is Record (Owner All_objects.Owner%Type
                          ,Object_name All_objects.Object_name%Type
                          ,object_type all_objects.object_type%type);

Type T_invalids Is Table Of R_invalids Index By Binary_integer;
g_invalids t_invalids;
```

Now we update the CHECK_INVALIDS procedure to fill the PL/SQL table:

```
PROCEDURE CHECK_INVALIDS(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
    select owner,object_name,object_type
    bulk collect into xxx_wf_dba.g_invalids
    from all_objects
    where status='INVALID';
    if xxx_wf_dba.g_invalids.count>0 then
      p_result:='Y';
      wf_engine.SetItemAttrNumber(itemtype=>p_item_type
                                ,itemkey=>p_item_key
                                ,aname=>'INVALID_CNTR'
                                ,avalue=>xxx_wf_dba.g_invalids.count);
    Else
      P_result:='N';
    End if;
  END IF;
```

```
END;  
/
```

This procedure checks for invalid objects and stores them in our PL/SQL table. If they are found, the result of the procedure will be 'Y', and the number of invalid objects will be stored in the item attribute 'INVALID_CNTR'.

This INVALID_CNTR will be used to limit the number of times the loop is executed. Of course we will only enter the loop when invalid objects are found.

The workflow engine uses a 'pivot' function to create a loop. This is the function that starts and ends the loop. It has 2 exits, one to enter the loop, and one to exit and continue with the rest of the process.

A seeded pivot function is delivered from the STANDARD itemtype: 'Loop Counter'. This uses a node attribute 'Loop Limit' to store the maximum number of times the loop can be executed. In our case we will assign 'INVALID_CNTR' to the 'Loop Limit'.

Of course the function also needs to keep track of the number of times the loop has actually been executed. For this the function creates a dynamic item attribute. This item attribute is called 'LOOP_COUNT:<process_activity>'. Where <process_activity is the process_activity_id of the 'Loop Counter' activity.

To read the PL/SQL table, we need an index pointer. We can use the LOOP_COUNT node attribute but since it has a dynamic name, it is not easy to use. Alternatively, we can use another item attribute or use a global PL/SQL variable. I will show both options in the code. First I start with a PL/SQL variable.

For this we add a variable 'g_counter NUMBER := 0;' to the package header. This variable will be increased on every call to our 'compile procedure'.

The compile procedure looks like this:

```
PROCEDURE compile(p_item_type IN VARCHAR2  
                ,p_item_key IN VARCHAR2  
                ,p_actid IN NUMBER  
                ,p_funcmode IN VARCHAR2  
                ,p_result OUT VARCHAR2) IS  
  
v_object_type varchar2(200);  
v_object_subtype varchar2(200);  
v_tab_object_type all_objects.object_type%type;  
  
PROCEDURE compile_pvt(p_owner IN VARCHAR2, p_object_type IN VARCHAR2, p_object_subtype IN  
VARCHAR2, p_object_name IN VARCHAR2) IS  
PRAGMA AUTONOMOUS_TRANSACTION;  
BEGIN  
    execute immediate 'alter '||p_object_type||' '||p_owner||'.'||p_object_name||  
        ' compile '||p_object_subtype;  
    commit;  
EXCEPTION  
    WHEN OTHERS THEN  
        IF sqlcode IN (-1031,-24344) THEN --Expected error messages on recompile.  
            NULL;  
        ELSE  
            RAISE;  
        END IF;  
END;  
END;
```

```

BEGIN
  IF p_funcmode='RUN' THEN
    xxx_wf_dba.g_counter:=xxx_wf_dba.g_counter+1;
    IF xxx_wf_dba.g_invalids(xxx_wf_dba.g_counter).object_type='PACKAGE BODY' THEN
      v_object_type:='PACKAGE';
      v_object_subtype:='BODY';
    ELSE
      v_object_type:=xxx_wf_dba.g_invalids(xxx_wf_dba.g_counter).object_type;
      v_object_subtype:=NULL;
    END IF;
    compile_pvt(xxx_wf_dba.g_invalids(xxx_wf_dba.g_counter).owner
      ,v_object_type
      ,v_object_subtype
      ,xxx_wf_dba.g_invalids(xxx_wf_dba.g_counter).object_name);
  END IF;
END;

```

The xxx_wf_dba.g_counter is increased, and then used as index pointer to the PL/SQL table. Notice that I used an autonomous transaction for the actual recompile. The reason for this is that the workflow engine does NOT accept commit or rollback in its activities. (And an alter is DDL which implicitly issues a commit) Only the calling session can issue a commit or rollback. In a straightforward process, the only issue will be that you can end up with an item that is executed completely and then rolled back to an illegal commit. (Which means you'll have to abort the item, or re-execute the offending activity.) But within a loop, the workflow engine will error out after a commit.

I leave it as an exercise to the reader to test the results of commits and or rollbacks in the workflow activities.

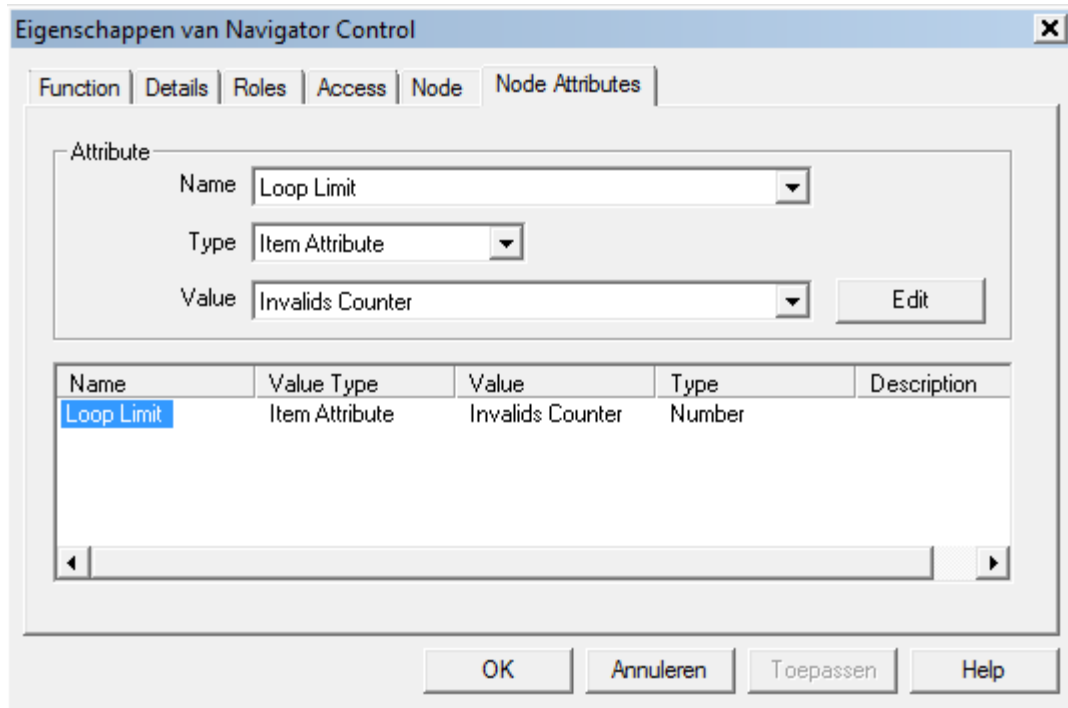
The paragraphs above will become more clear when we start building our process. First we add the 'Loop Counter':

The screenshot shows the 'Eigenschappen van Navigator Control' dialog box. It has tabs for 'Function', 'Details', 'Roles', 'Access', 'Node', and 'Node Attributes'. The 'Details' tab is active. The fields are as follows:

- Item Type: Standard (dropdown menu)
- Internal Name: LOOPCOUNTER (dropdown menu)
- Display Name: Loop Counter (text field)
- Description: Counts the number of times a loop has been travers (text field)
- Icon: ADD.ICO (dropdown menu) with a green plus sign icon and a 'Browse' button
- Function Name: WF_STANDARD.LOOPCOUNTER (text field)
- Function Type: PL/SQL (dropdown menu)
- Result Type: Loop Counter (dropdown menu)
- Cost: 0.01 (text field)

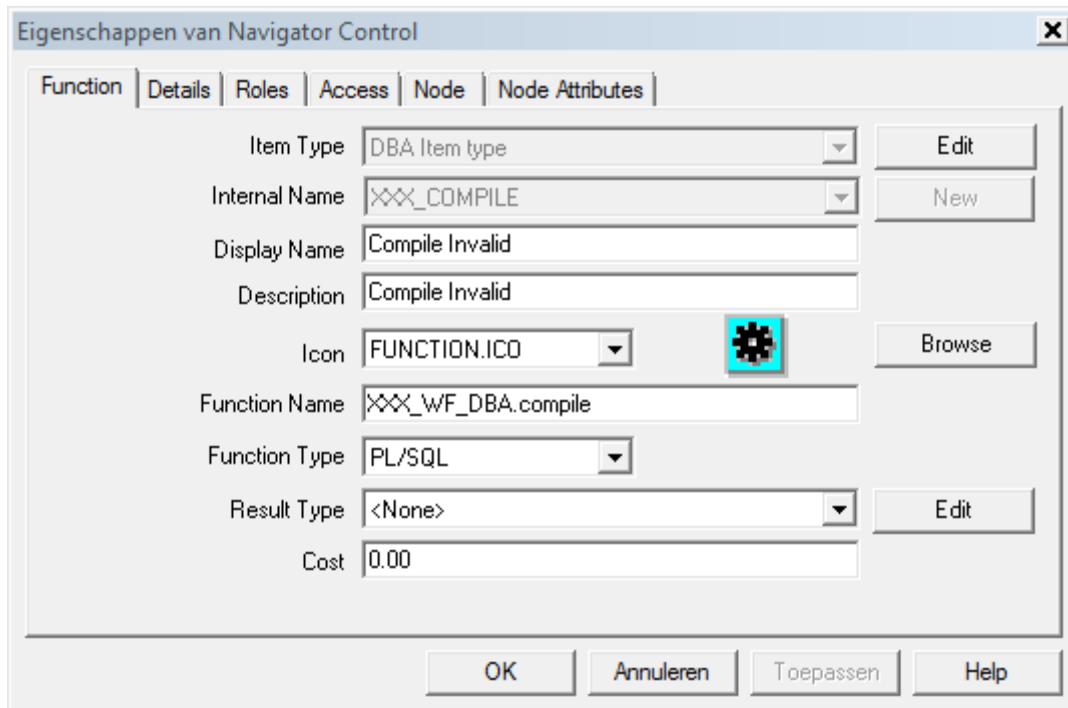
Buttons at the bottom: OK, Annuleren, Toepassen, Help.

The 'Loop Limit' is defined as:



As usual you can also use a constant.

Then we add the compile procedure:



Then finally we build the process:


```

        and wat.language='US')
where wias.item_type='DBA_TYPE'
and wias.item_key='7'
order by wias.begin_date,wias.execution_time;

```

| ITEM_KEY | DISPLAY_NAME | FUNCTION | BEGIN_DAT | END_DATE | STATUS | RESULT_CODE |
|----------|-----------------|-------------------------|-----------|----------|----------|-------------|
| 7 | Loop Counter | WF_STANDARD.LOOPCOUNTER | 13-10-09 | 13-10-09 | COMPLETE | LOOP |
| 7 | Compile Invalid | XXX_WF_DBA.compile | 13-10-09 | 13-10-09 | COMPLETE | |
| 7 | Loop Counter | WF_STANDARD.LOOPCOUNTER | 13-10-09 | 13-10-09 | COMPLETE | LOOP |
| 7 | Compile Invalid | XXX_WF_DBA.compile | 13-10-09 | 13-10-09 | COMPLETE | |
| 7 | Loop Counter | WF_STANDARD.LOOPCOUNTER | 13-10-09 | 13-10-09 | COMPLETE | LOOP |
| 7 | Compile Invalid | XXX_WF_DBA.compile | 13-10-09 | 13-10-09 | COMPLETE | |
| 7 | Loop Counter | WF_STANDARD.LOOPCOUNTER | 13-10-09 | 13-10-09 | COMPLETE | LOOP |
| 7 | Compile Invalid | XXX_WF_DBA.compile | 13-10-09 | 13-10-09 | COMPLETE | |
| 7 | Loop Counter | WF_STANDARD.LOOPCOUNTER | 13-10-09 | 13-10-09 | COMPLETE | LOOP |
| 7 | Compile Invalid | XXX_WF_DBA.compile | 13-10-09 | 13-10-09 | COMPLETE | |

Now it is clear that the 'Loop Counter' has first returned 'LOOP' until the 'Loop Limit' was reached. Then it returned 'EXIT'.

We can verify the number of times the 'Compile Invalid' activity was called. First from the PL/SQL variable:

```

begin
dbms_output.put_line(xxx_wf_dba.g_counter);
end;

```

When we look at the item attributes, we see that the LOOP_COUNT attribute has been reset to 0 again. This is of course necessary, since the process can enter the loop again later.

```

select item_key,name,text_value,number_value
from wf_item_attribute_values
where item_type='DBA_TYPE'
and item_key='7';

```

| ITEM_KEY | NAME | TEXT_VALUE | NUMBER_VALUE |
|----------|-------------------|---|--------------|
| 7 | #SCHEMA | APPS | |
| 7 | .ADMIN_KEY | 222486919219571661780940100168177866039 | |
| 7 | .MONITOR_KEY | 130597263481821240071415478765141466910 | |
| 7 | INSTANCE_NAME | VIS | |
| 7 | INVALID_CNTR | | 20 |
| 7 | LOOP_COUNT:788217 | | 0 |

If you want to run the process a second time, you'll first need to reset the PL/SQL variable again. You can avoid this by using an item attribute.

First we define the attribute:

The screenshot shows a dialog box titled "Eigenschappen van Navigator Control". It has two tabs: "Attribute" and "Access". The "Attribute" tab is selected. The fields are as follows:

- Item Type: DBA Item type
- Internal Name: COUNTER
- Display Name: Loop Counter
- Description: Loop Counter
- Type: Number (dropdown menu)
- Format: (empty text box)
- Default section:
 - Type: Constant (dropdown menu)
 - Value: 1

At the bottom, there are four buttons: OK, Annuleren, Toepassen, and Help.

Whether to start with value 1 and update at the end of the processing, or to start with value 0 and update before the processing does not matter in this case. However it is an important choice to make, when error handling is involved and the counter is also used in other functions.

Now we change the procedure to read the item attribute:

```

PROCEDURE compile(p_item_type IN VARCHAR2
                 ,p_item_key IN VARCHAR2
                 ,p_actid IN NUMBER
                 ,p_funcmode IN VARCHAR2
                 ,p_result OUT VARCHAR2) IS

v_object_type varchar2(200);
v_object_subtype varchar2(200);
v_tab_object_type all_objects.object_type%type;
v_counter number;

PROCEDURE compile_pvt(p_owner IN VARCHAR2, p_object_type IN VARCHAR2, p_object_subtype IN
VARCHAR2, p_object_name IN VARCHAR2) IS
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    execute immediate 'alter '||p_object_type||' '||p_owner||'.'||p_object_name||
    ' compile '||p_object_subtype;
    commit;
EXCEPTION
    WHEN OTHERS THEN
        IF sqlcode IN (-1031,-24344) THEN --Expected error messages on recompile.
            NULL;
        ELSE
            --NULL;
            RAISE;
        END IF;
END;

BEGIN
    IF p_funcmode='RUN' THEN
        v_counter := wf_engine.GetItemAttrNumber(itemtype=>p_item_type
                                                ,itemkey=>p_item_key
                                                ,aname=>'COUNTER');
--
        xxx_wf_dba.g_counter:=xxx_wf_dba.g_counter+1;
        IF xxx_wf_dba.g_invalids(v_counter).object_type='PACKAGE BODY' THEN
            v_object_type:='PACKAGE';
            v_object_subtype:='BODY';

```

```

ELSE
  v_object_type:=xxx_wf_dba.g_invalids(v_counter).object_type;
  v_object_subtype:=NULL;
END IF;
compile_pvt(xxx_wf_dba.g_invalids(v_counter).owner
            ,v_object_type
            ,v_object_subtype
            ,xxx_wf_dba.g_invalids(v_counter).object_name);
wf_engine.SetItemAttrNumber(itemtype=>p_item_type
                            ,itemkey=>p_item_key
                            ,aname=>'COUNTER'
                            ,avalue=>v_counter+1);

END IF;
END;

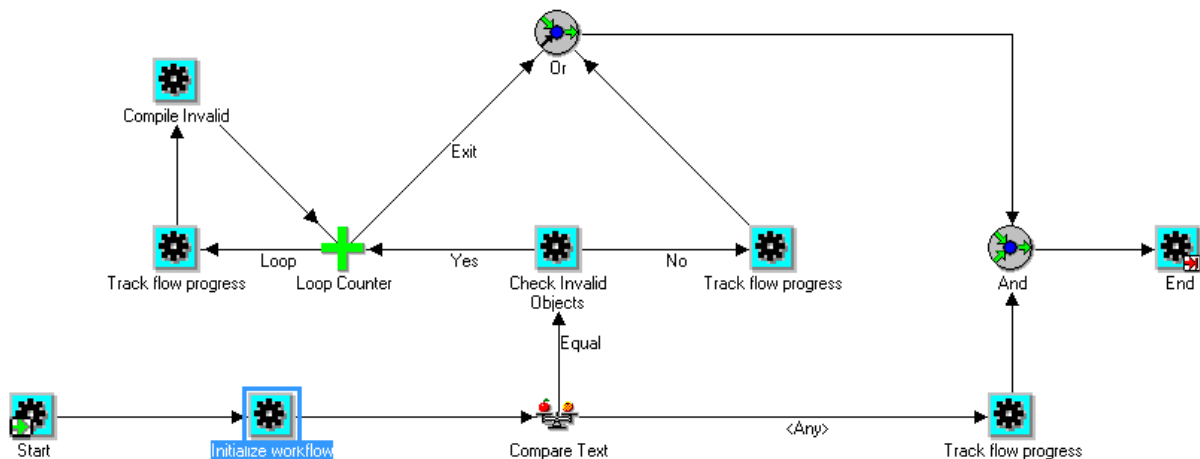
```

When we execute this, the loop will also be traversed the required number of times. But now we can follow the number of executions from the item attribute:

| ITEM_KEY | NAME | TEXT_VALUE | NUMBER_VALUE |
|----------|-------------------|---|--------------|
| 7 | #SCHEMA | APPS | |
| 7 | .ADMIN_KEY | 209173792773616004683085036749783207440 | |
| 7 | .MONITOR_KEY | 273785298922858400646744615769101227691 | |
| 7 | COUNTER | | 21 |
| 7 | INSTANCE_NAME | VIS | |
| 7 | INVALID_CNTR | | 20 |
| 7 | LOOP_COUNT:788355 | | 0 |

Since this time, I increase the counter after processing, it shows 21. The 'Loop Counter' exited after the 20th execution.

Now it's time to have some fun with some other features of loops. To track the progress, I added an extra 'Track Flow Progress' function in the loop.



We will now zoom in on the 'Pivot' activity. You can see that it will be the first activity that is called for the second time. In this case, that is because it is part of the loop. But an activity can be called multiple times in other situations too. For example when different tracks of a process are joined together again. Instead of putting the 'And' or 'Or' functions, it might be needed to have the next part of the process executed just once. Another possible situation is where part of the process has to be re-done with different input parameters.

We are going to see the possibilities of these now. The behavior of the pivot function is controlled by a property on the 'Details' tab: 'On Revisit'. This property can be set to: 'Loop', 'Reset' or 'Ignore'.

You will see that the 'Loop Counter' defaults to 'Loop'. That means that it will re-execute all the activities following the pivot function. It makes a loop, as we have seen above.

When you set the property to 'Reset', it will try to re-do the activities following the pivot-function. Let's take a look at the results of the above process with the 'On Revisit' for the 'Loop Counter' to 'Reset':

| ITEM_KEY | DISPLAY_NAME | FUNCTION | BEGIN_DAT | END_DATE | STATUS | RESULT_CODE |
|----------|-----------------------|--------------------------------|-----------|----------|----------|-------------|
| 8 | DBA Main Process | | 13-10-09 | 13-10-09 | COMPLETE | #NULL |
| 8 | Start | START | 13-10-09 | 13-10-09 | COMPLETE | #NULL |
| 8 | Initialize workflow | XXX_WF_DBA.init | 13-10-09 | 13-10-09 | COMPLETE | COMPLETE |
| 8 | Compare Text | WF_STANDARD.COMPARE | 13-10-09 | 13-10-09 | COMPLETE | EQ |
| 8 | Track flow progress | XXX_WF_DBA.track_flow_progress | 13-10-09 | 13-10-09 | COMPLETE | |
| 8 | Check Invalid Objects | XXX_WF_DBA.check_invalids | 13-10-09 | 13-10-09 | COMPLETE | Y |
| 8 | Loop Counter | WF_STANDARD.LOOPCOUNTER | 13-10-09 | 13-10-09 | COMPLETE | EXIT |
| 8 | Or | WF_STANDARD.ORJOIN | 13-10-09 | 13-10-09 | COMPLETE | #NULL |
| 8 | And | WF_STANDARD.ANDJOIN | 13-10-09 | 13-10-09 | COMPLETE | #NULL |
| 8 | End | END | 13-10-09 | 13-10-09 | COMPLETE | #NULL |

So far it seems to have executed correctly. Also the 'Counter' item attribute shows that the 'Compile' function has been called 20 times:

| ITEM_KEY | NAME | NUMBER_VALUE |
|----------|-------------------|--------------|
| 8 | COUNTER | 21 |
| 8 | INVALID_CNTR | 20 |
| 8 | LOOP_COUNT:788425 | 0 |

*) removed some irrelevant records.

But the surprise comes when we check the Track flow table:

```
select *
from xxx_track_flow
where item_type='DBA_TYPE'
and item_key='8'
```

| ID | ITEM_TYPE | ITEM_KEY | ACTIVITY_NAME | VERSION | PROCESS_NAME | INSTANCE | INSTANCE_LABEL | FUNCMODE |
|-----|-----------|----------|---------------|---------|------------------|----------|----------------|----------|
| 126 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788433 | TRACK_FLOW-2 | RUN |
| 127 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | RUN |
| 128 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | CANCEL |
| 129 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | RUN |
| 130 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | CANCEL |
| 131 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | RUN |
| 132 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | CANCEL |
| 133 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | RUN |
| 134 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | CANCEL |
| 135 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | RUN |
| 136 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | CANCEL |
| 137 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | RUN |
| 138 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | CANCEL |
| 139 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | RUN |
| 140 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | CANCEL |
| 141 | DBA_TYPE | 8 | TRACK_FLOW | 12 | DBA_MAIN_PROCESS | 788439 | TRACK_FLOW | RUN |

This is one of the situations where the 'Funcmode' plays a role. Before it was always called with value 'RUN'. But with the 'On Revisit' property set to 'Reset', it is executing the activities after the pivot function twice. The first time, the loop is executed in 'RUN' mode. On the second pass, the activities are first called with a mode 'CANCEL'. This enables the activities to un-do their actions from the first pass. After that, the activities are called again with a mode 'RUN'. You can imagine this to be useful in situations where activities have to be executed with the latest input parameters.

Now it also becomes clear why the workflow engine did not allow us to use a commit in the procedure. Since that might prevent the activity from un-doing its work.

Now let's change the 'On Revisit' to 'Ignore':

| ITEM_KEY | DISPLAY_NAME | FUNCTION | BEGIN_DAT | END_DATE | STATUS | RESULT_CODE |
|----------|-----------------------|--------------------------------|-----------|----------|----------|-------------|
| 9 | DBA Main Process | | 13-10-09 | | ACTIVE | #NULL |
| 9 | Start | START | 13-10-09 | 13-10-09 | COMPLETE | #NULL |
| 9 | Initialize workflow | XXX_WF_DBA.init | 13-10-09 | 13-10-09 | COMPLETE | COMPLETE |
| 9 | Compare Text | WF_STANDARD.COMPARE | 13-10-09 | 13-10-09 | COMPLETE | EQ |
| 9 | Track flow progress | XXX_WF_DBA.track_flow_progress | 13-10-09 | 13-10-09 | COMPLETE | |
| 9 | And | WF_STANDARD.ANDJOIN | 13-10-09 | | WAITING | |
| 9 | Check Invalid Objects | XXX_WF_DBA.check_invalids | 13-10-09 | 13-10-09 | COMPLETE | Y |
| 9 | Loop Counter | WF_STANDARD.LOOPCOUNTER | 13-10-09 | 13-10-09 | COMPLETE | LOOP |
| 9 | Track flow progress | XXX_WF_DBA.track_flow_progress | 13-10-09 | 13-10-09 | COMPLETE | |
| 9 | Compile Invalid | XXX_WF_DBA.compile | 13-10-09 | 13-10-09 | COMPLETE | |

It's clear that this is not a suitable option for building a loop. Let's look at the 'Track flow' table:

| ID | ITEM_TYPE | ITEM_KEY | ACTIVITY_NAME | VERSION | PROCESS_NAME | INSTANCE | INSTANCE_LABEL | FUNCMODE |
|-----|-----------|----------|---------------|---------|------------------|----------|----------------|----------|
| 167 | DBA_TYPE | 9 | TRACK_FLOW | 13 | DBA_MAIN_PROCESS | 788503 | TRACK_FLOW-2 | RUN |
| 168 | DBA_TYPE | 9 | TRACK_FLOW | 13 | DBA_MAIN_PROCESS | 788509 | TRACK_FLOW | RUN |

The loop was executed just once. After that, the 'Loop Counter' did not execute again. The workflow engine stopped executing the item there.

This might not make sense in this case. But imagine where multiple tracks lead into an activity, where it should be executed just once. For example an approval workflow, that was split into multiple different tracks for different approval methods. And only the first one returning needs to be handled while the others should be ignored. Just make sure that the workflow engine is able to reach an 'End' node when you set the 'On Revisit' to 'Ignore'.

With that we finish part 3 of our series. In the next part, we'll look at the workflow controls like 'Defer' and 'Block'.