

Workflow for eBS DBA's (part 4)

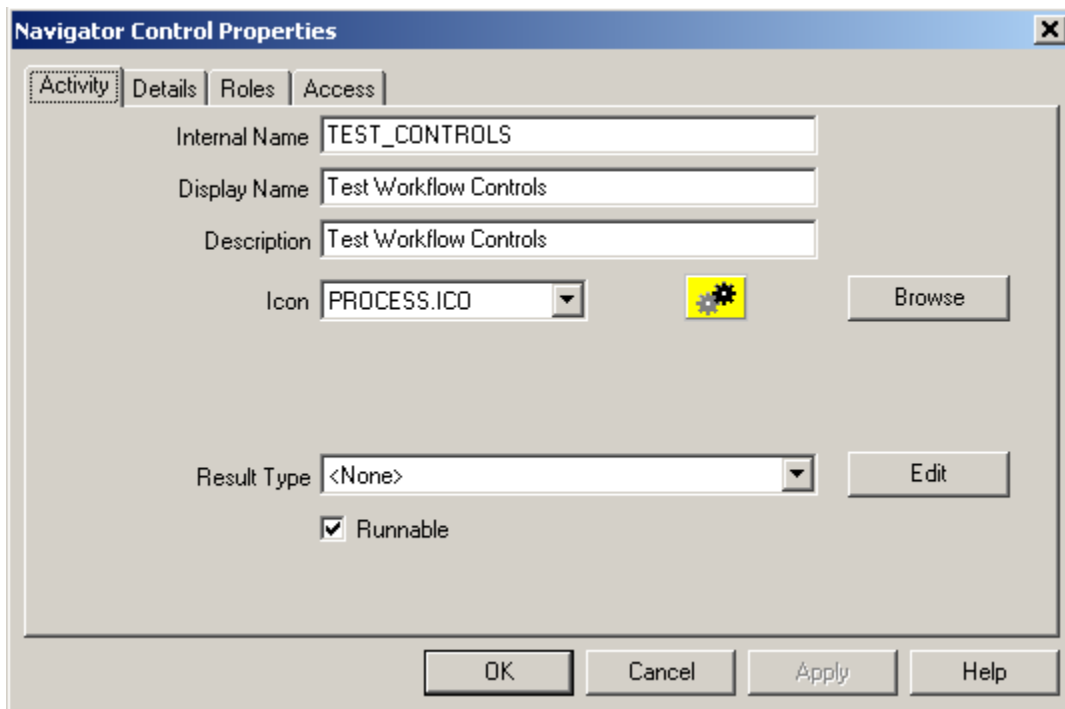
In this part of our series (previous parts to be found [here](#), [here](#) and [here](#)) we will look into controlling the workflow items. There are a number of ways to hold, or delay an item. We can also let the item be taken over by another session, and of course we can abort the item completely. We are going through all of these 'Controls' one by one.

The first that we are going to look into is the 'Defer'. You'll remember that the session that starts an item waits for it to complete (or error out). Especially for long-running functions this is not desirable. Therefore Oracle delivers the 'Workflow Background Process'. It is available as a concurrent request in eBS (from the system administrator responsibility). But in this article, we'll start the corresponding API.

First we are going to see how it works functionally. Then we'll see how it works on the database level. And finally, we look into costing your functions so that your items can be handled by multiple background engines.

To transfer the execution of your item to the background engine, you can call the 'Defer' function. Available from the 'STANDARD' itemtype.

Let's first build a small process:



The screenshot shows the 'Navigator Control Properties' dialog box with the 'Activity' tab selected. The fields are as follows:

Field	Value
Internal Name	TEST_CONTROLS
Display Name	Test Workflow Controls
Description	Test Workflow Controls
Icon	PROCESS.ICO
Result Type	<None>
Runnable	<input checked="" type="checkbox"/>

Buttons: OK, Cancel, Apply, Help, Browse, Edit.

Then our 'Defer' function:

Navigator Control Properties

Function Details Roles Access Node Node Attributes

Item Type Standard Edit

Internal Name DEFER New

Display Name Defer Thread

Description Defers the thread to a background engine

Icon TRANSFRM.ICO Browse

Function Name WF_STANDARD.DEFER

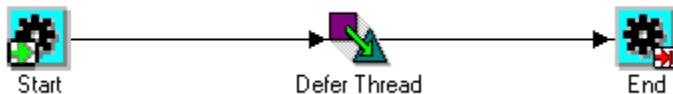
Function Type PL/SQL

Result Type <None> Edit

Cost 0.00

OK Cancel Apply Help

And we get a process like this:



If you're interested in how the process itself is stored in the WF_* tables, I refer you to [part 2](#). Now we are only going to see how an item of this process will behave.

```

Begin
Wf_engine.Launchprocess(Itemtype=>'DBA_TYPE',Itemkey=>'10',Process=>'TEST_CONTROLS');
end;

select wias.item_key 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 status
, Wias.Activity_result_code result
, wias.process_activity
, wias.outbound_queue_id
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='10'
  
```

```
order by wias.begin_date,wias.execution_time;
```

ITE	DISPLAY_NAME	FUNCTION	PROCESS_A	BEGIN_DAT	END_DATE	STATUS	RESULT	OUTBOUND_QUEUE_ID
10	Test Workflow Controls		788609	18-10-09		ACTIVE	#NULL	
10	Start	START	788648	18-10-09	18-10-09	COMPLETE	#NULL	
10	Defer Thread	WF_STANDARD.DEFER	788646	18-10-09		DEFERRED	#NULL	686F85A5874F4A818B4BB62EC7920520

First thing you'll notice is that control was returned to the calling session, but the workflow item did not complete yet. It executed the 'Defer Thread' function and then returned. The Activity_status is now 'DEFERRED', and an 'OUTBOUND_QUEUE_ID' has been assigned.

The status 'DEFERRED' means that the item is waiting for processing by a background engine. But before we call the background engine, we'll look at the queuing mechanism.

When the workflow engine finds a function with result_code 'DEFERRED' or actually 'wf_engine.eng_deferred', it knows the item needs to be deferred. It will then update the status of the item_activity and then put a message on an 'Advanced Queuing' or AQ-queue. In this case, the queue is 'WF_DEFERRED_QUEUE_M'. Do not mistake this queue for the 'WF_DEFERRED' queue. The 'WF_DEFERRED' queue is used by the workflow listeners and the 'Business Event System' (BES). We'll discuss them later.

On the 'WF_DEFERRED_QUEUE_M' a message is enqueued with a 'Correlation Identifier' (CORRID) of 'APPS' concatenated with the item_type. The payload for the message consists of item_type, item_key and process_activity.

The AQ-queues can be queried by the underlying queue table or the queueing view. The view will return translated values for some of the columns.

Let's see what has been queued for our item. The key is of course the 'OUTBOUND_QUEUE_ID' from 'WF_ITEM_ACTIVITY_STATUSES':

```
select queue
,      corr_id
,      msg_priority priority
,      msg_state
,      enq_time
,      def.user_data.itemtype itemtype
,      def.user_data.itemkey itemkey
,      def.user_data.actid actid
,      consumer_name
from applsys.AQ$WF_DEFERRED_TABLE_M def
where msg_id=(select outbound_queue_id
               from wf_item_activity_statuses
               where item_type='DBA_TYPE'
               and item_key='10'
               and activity_status='DEFERRED');
```

QUEUE	CORR_ID	PRIORITY	MSG_STATE	ENQ_TIME	ITEMTYPE	ITEMKEY	ACTID	CONSUMER_NAME
WF_DEFERRED_QUEUE_M	APPSDBA_TYPE	1	READY	18-10-09	DBA_TYPE	10	788646	APPS

The columns will be self-explanatory. There are more columns in the view. But these are not relevant for us. As we can see, the message is ready to be picked up. So let's run the background engine now.

```
Begin
  Wf_engine.Background(Itemtype=>'DBA_TYPE'
                      ,Process_deferred=>True);
```

End;

The background engine has more parameters. We'll see the min- and maxthreshold when discussing costing. There are also parameters for processing time-out and stuck. Both will be discussed at the end of this article. The same parameters exist for the concurrent program.

Now that the workflow engine has run, we can look at the status of the item again.

KEY	DISPLAY_NAME	FUNCTION	PROCESS_A	BEGIN_DATE	END_DATE	STATUS	RESULT	OUTBOUND_QU
10	Test Workflow Controls		788609	18-10-09 15:54:49	18-10-09 16:43:30	COMPLETE	#NULL	
10	Start	START	788648	18-10-09 15:54:50	18-10-09 16:43:30	COMPLETE	#NULL	
10	Defer Thread	WF_STANDARD.DEFER	788646	18-10-09 16:43:30	18-10-09 16:43:30	COMPLETE	#NULL	
10	End	END	788650	18-10-09 16:43:30	18-10-09 16:43:30	COMPLETE	#NULL	

The `outbound_queue_id` has been nullified now. And the item is complete now.

It is not very clear from the above example, but be aware that the background engine is issuing a commit on finishing an item.

Also I changed my client to show the entire timestamp. You'll notice that the 'Defer Thread' has a `begin_time` of 16:43:30. This is actually not the time it was first run, but the time the background engine was run.

This is due to the way the background engine works. When the background engine is run, it just re-executes the function that is deferred.

The `WF_STANDARD.DEFER` function first checks if it's process activity already has a `result_code`. If not, it returns a `result_code` 'wf_engine.eng_defer', which prompts the workflow engine to defer the item. The second time the 'WF_STANDARD.DEFER' is called it will find an existing `result_code` and return a 'COMPLETE'.

This is something you should be aware of, when you create your own functions that should be deferrable.

Costing

Let's take a look at the workflow costing model. You'll have noticed the field 'Cost' in the workflow builder on the functions. As you saw before, the 'Defer' function enables you to let the background engine run long-running or resource intensive jobs. But in general, a background engine will pick up all deferred items for a specified `item_type`.

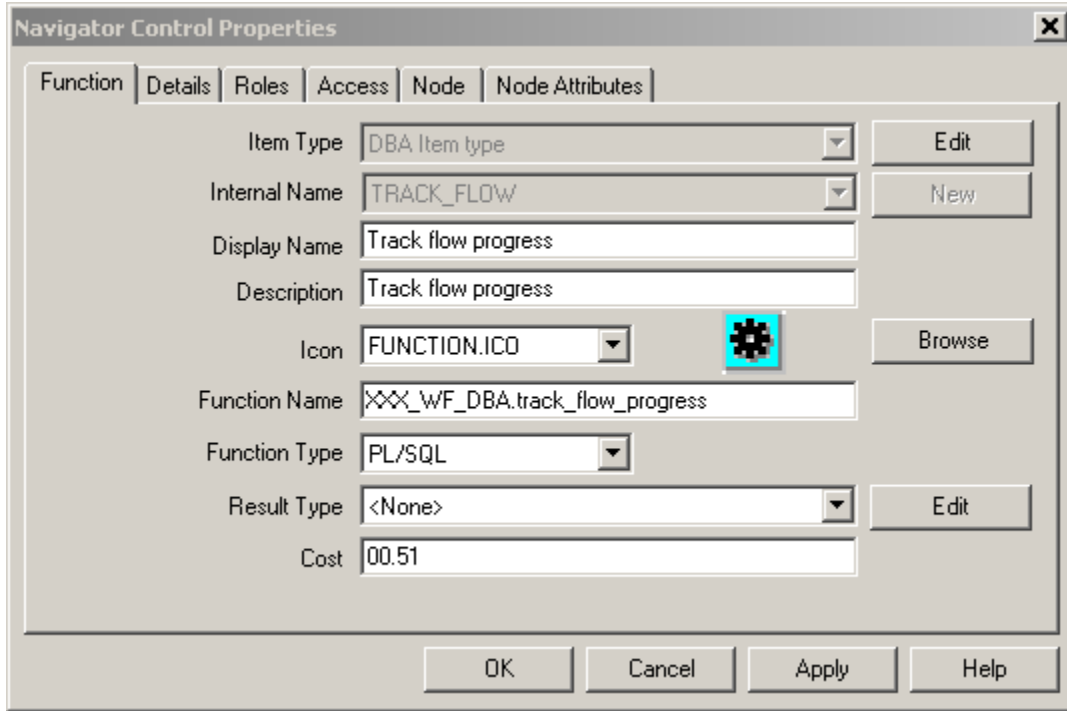
It might be necessary to make a further graduation in this. Either to let the heaviest of functions run only at certain times, or because you want to run background engines running parallel without concurrency waits between them. The whole workflow engine is transaction bound. So do not worry that multiple background engines will corrupt each other's data. However buffer busy wait or enqueue waits can occur when running multiple background engines.

To further specialize background engines you can use the 'Cost'. Cost is meant as an indicator for the runtime of the function in hundredths of seconds. It can be set to a value between 0.00 and 1,000,000.00 (which leaves a range of a hundred million).

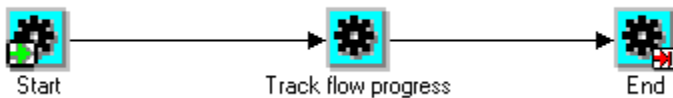
By default any function with a cost higher than 00.50 (>00.50) will be deferred by the workflow engine. And the background engine will only pick up items with a cost between its minimum and maximum threshold.

Let's see how this works by replacing the 'Defer Thread' with our 'Track Flow Progress' function. (Check [part 3](#) for the source of the function).

We set the cost of the function to 00.51:



And of course put it in our process:



Now when we run the item, we can see it has been deferred:

KEY	DISPLAY_NAME	FUNCTION	PROCESS	BEGIN_DAT	END_DATE	STATUS	OUTBOUND_Q
11	Test Workflow Controls		788609	18-10-09		ACTIVE	
11	Start	START	788800	18-10-09	18-10-09	COMPLETE	
11	Track flow progress	XXX_WF_DBA.track_flow_progress	788804	18-10-09		DEFERRED	55631752A867410EAD76625CB95B0649

Now we can already see the minthreshold and maxthreshold at work. When we run the background engine with

```
maxthreshold=>50
or
minthreshold=>52
```

the item will not be processed. Note that the maxthreshold tests for cost <= maxthreshold. Minthreshold tests for cost > minthreshold. Both parameters are set in hundredths of a second. Or a hundred-fold the value in the workflow builder.

When we run the background engine without any threshold, or with the above values, the item will be processed.

After running the background engine, we can also see the double execution in 'XXX_TRACK_FLOW':

ID	ITEM_TYPE	ITEM_KEY	ACTIVITY_NAME	VERSION	PROCESS_NAME	INSTANCE	INSTANCE_LABEL	FUNCMODE
187	DBA_TYPE	11	TRACK_FLOW	19	TEST_CONTROLS	789035	TRACK_FLOW	RUN
188	DBA_TYPE	11	TRACK_FLOW	19	TEST_CONTROLS	789035	TRACK_FLOW	RUN

I leave it as an exercise to the reader to test with the different values of the thresholds and cost.

A last point to mention with the cost is the default value of 50, for the workflow engine to defer an activity. This can be adjusted in the workflow engine. If you call

```
Wf_engine.threshold := n;
```

Before the start of an item, the workflow engine will use this value as the threshold for deferral.

Then some words on issues with the background engine or the 'Workflow Background Process' in eBS.

One of the most common problems is a long runtime for the process. This can be caused by a number of issues. The most common ones are:

- No purging on the workflow tables
- Processing delaying tests in a loop
- High QMON activity caused by oversized IOT's

For the first issue. If you don't run the 'Purge Obsolete Workflow Runtime Data' process regularly, it's a good time to start running it. This program purges the completed items from the database. If it is not run regularly, the workflow tables will bloat and cause all workflow processing to go slower over time. We will look into the purging process later in this series.

The second issue can be caused by testing for a certain condition and if it is not true, deferring the item. If there is no further delay, the background engine will continue testing the condition and deferring the item. Ultimately, the background engine will be looping on this test.

Another case where this can occur is on very low time-out values. Time-outs will be discussed later in this article. But when set very low, they can also cause the background engine to loop on one item.

Finally, you might notice high CPU or I/O usage for the QMON process (QMNx). In that case, it might be needed to coalesce the IOT's for AQ. See Metalink note 271855.1 for more details about this.

For us, it's time to look at the

BLOCK

When an item needs to put on hold for an undetermined period of time, the 'BLOCK' function from the 'STANDARD' itemtype can be used. It holds the processing of the item until the 'WF_ENGINE.CompleteActivity' function is called.

In most cases, the Business Event System (BES) can be used for this too. But 'BLOCK' is very straightforward and often used.

Let's change our process to include one:

Navigator Control Properties

Function Details Roles Access Node Node Attributes

Item Type Standard Edit

Internal Name BLOCK New

Display Name Block

Description Wait for external completion.

Icon STOP.ICO Browse

Function Name WF_STANDARD.BLOCK

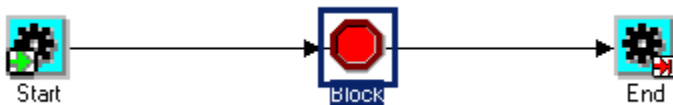
Function Type PL/SQL

Result Type <None> Edit

Cost 0.01

OK Cancel Apply Help

And the process:



After we start an item, we see the following:

KEY	DISPLAY_NAME	FUNCTION	PROCESS_A	BEGIN_DAT	END_DATE	STATUS	RESULT_CODE	OUTBOUND_QUEUE_ID
12	Test Workflow Controls		788609	18-10-09		ACTIVE	#NULL	
12	Start	START	789108	18-10-09	18-10-09	COMPLETE	#NULL	
12	Block	WF_STANDARD.BLOCK	789112	18-10-09		NOTIFIED		

The item is still active. And the 'Block' has an activity_status of 'NOTIFIED'. There is no 'OUTBOUND_QUEUE_ID'.

The item will be resumed when a session calls 'WF_ENGINE.CompleteActivity'.

```

Begin
  Wf_engine.Completeactivity(Itemtype=>'DBA_TYPE'
    ,Itemkey=>'12'
    ,Activity=>'TEST_CONTROLS:BLOCK'
    ,result=>wf_engine.eng_null);
end;
  
```

The activity consists of the process concatenated with a colon to the instance-label of the activity. The result has to be a valid result code for the activity. In our case, there is no Result Type defined, so we returned an empty result.

This time, the activity is not executed a second time. The workflow engine now sets the activity to 'COMPLETED' and continues with the next process activity.

KEY	DISPLAY_NAME	FUNCTION	PROCESS_A	BEGIN_DATE	END_DATE	STATUS	RESULT	OUTBOUND_QU
12	Test Workflow Controls		788609	18-10-09 18:50:35	18-10-09 18:50:40	COMPLETE	#NULL	
12	Start	START	789108	18-10-09 18:50:35	18-10-09 18:50:35	COMPLETE	#NULL	
12	Block	WF_STANDARD.BLOCK	789112	18-10-09 18:50:35	18-10-09 18:50:40	COMPLETE	#NULL	
12	End	END	789110	18-10-09 18:50:40	18-10-09 18:50:40	COMPLETE	#NULL	

As you can see, the 'Block' function is very straightforward. It holds the item until somebody calls the 'CompleteActivity' function. Whether the 'CompleteActivity' function will ever be called is decided entirely outside the workflow system. This is not always desirable, because an error outside the workflow system may cause the item to be on hold indefinitely. Sometimes you want to escalate things when an appropriate response does not arrive in time.

For this, we have a time-out function in the workflow engine.

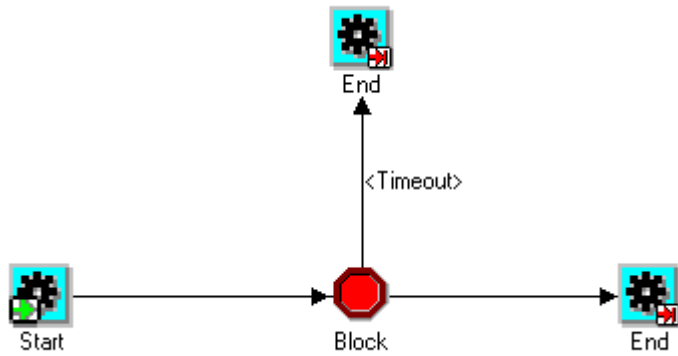
On the 'Node' tab of the function, a time-out is available.

The screenshot shows the 'Navigator Control Properties' dialog box with the 'Node' tab selected. The 'Label' field contains 'BLOCK'. The 'Start/End' dropdown is set to 'Normal'. The 'Comment' field is empty. The 'Timeout' section has a 'Type' dropdown set to 'Relative Time' and a 'Value' field with three input boxes: '0' for days, '0' for hours, and '5' for minutes. The 'Performer' section has a 'Type' dropdown set to 'Constant' and a 'Value' dropdown set to '<None>', with an 'Edit' button next to it. At the bottom are buttons for 'OK', 'Cancel', 'Apply', and 'Help'.

It can be set either to a relative time, or the value of an item attribute. The relative time is shown above. This is counted from the begin_date of the process activity. When an item attribute is used it is set to a fixed date-value, including timestamp.

Note that the time-out is available for ALL functions. Not just the 'Block'. This is just a common feature to use, since the completion is external. Functions that are handled internally usually don't require the time-out.

Of course we also have a result-code for a timed-out activity:



Now when we run the process, we see the expected result, with the block 'NOTIFIED':

KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	DUE_DATE	STATUS	RESULT
13	Test Workflow Controls		25-10-09 12:33:40			ACTIVE	#NULL
13	Start	START	25-10-09 12:33:40	25-10-09 12:33:40		COMPLETE	#NULL
13	Block	WF_STANDARD.BLOCK	25-10-09 12:33:40		25-10-09 12:38:40	NOTIFIED	

I added the column 'DUE_DATE' from WF_ITEM_ACTIVITY_STATUSES to the query. Here we can see that it is set for the 'Block', with a timestamp 5 minutes after the begin_date.

To continue the item after a time-out, we have to run a background engine with a parameter 'processed_timeout=>TRUE'. (Or a 'Y' for the time-out parameter on the concurrent program).

```

Begin
Wf_engine.Background(Itemtype=> 'DBA_TYPE',process_Timeout=>True);
end;
  
```

And the result:

KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	DUE_DATE	STATUS	RESULT
13	Test Workflow Controls		25-10-09 12:33:40	25-10-09 12:39:16		COMPLETE	#NULL
13	Start	START	25-10-09 12:33:40	25-10-09 12:33:40		COMPLETE	#NULL
13	Block	WF_STANDARD.BLOCK	25-10-09 12:33:40	25-10-09 12:39:16	25-10-09 12:38:40	COMPLETE	#TIMEOUT
13	End	END	25-10-09 12:39:16	25-10-09 12:39:16		COMPLETE	#NULL

The background process completed the 'Block' with a result_code of '#TIMEOUT'. Note how the begin_date has not changed, like when a 'DEFERRED' activity is processed. This time, the activity was just completed and not re-executed.

So what happens when the 'CompleteActivity' is executed after the time-out:

```

Begin
  Wf_engine.Completeactivity(Itemtype=>'DBA_TYPE'
    ,Itemkey=>'13'
    ,Activity=>'TEST_CONTROLS:BLOCK'
    ,result=>wf_engine.eng_null);
end;
  
```

```

ORA-20002: 3133: Activity instance 'TEST_CONTROLS:BLOCK' is not a notified activity for item 'DBA_TYPE/13'.
ORA-06512: at "APPS.WF_ENGINE", line 5688
ORA-06512: at line 2
  
```

It returns a workflow error, saying the activity to be completed is not in 'Notified' state. Usually this error indicates that the activity has either been completed by another process, or it has run into a time-out and has been processed by a background engine.

Of course all code that is completing these activities need to have error-handling in place for this.

Let's take a look at sample code that handles this error:

```
declare
v_errordname varchar2(30);
v_errormsg varchar2(2000);
v_errorstack varchar2(32000);
invalid_action EXCEPTION;
PRAGMA EXCEPTION_INIT(invalid_action, -20002);
begin
    wf_engine.completeactivity(itemtype=>'DBA_TYPE'
                              ,itemkey=>'21'
                              ,activity=>'TEST_CONTROLS:BLOCK'
                              ,result=>wf_engine.eng_completed);
exception
    when invalid_action THEN
        wf_core.get_error(v_errordname,v_errormsg,v_errorstack);
        dbms_output.put_line(v_errordname);
        dbms_output.put_line(v_errormsg);
        dbms_output.put_line(v_errorstack);
end;
/
```

This code just returns the error to the output buffer. In production code, the error-handler should establish whether the activity has timed-out or not. This can be done by calling:

```
Wf_item_activity_status.Result(Itemtype => v_itemtype
                              ,Itemkey  => v_itemkey
                              ,Actid    => v_actid
                              ,Status   => v_status
                              ,Result   => v_result);
```

Where v_actid is the process_activity_id and v_status / v_result are the activity_status and activity_result_code.

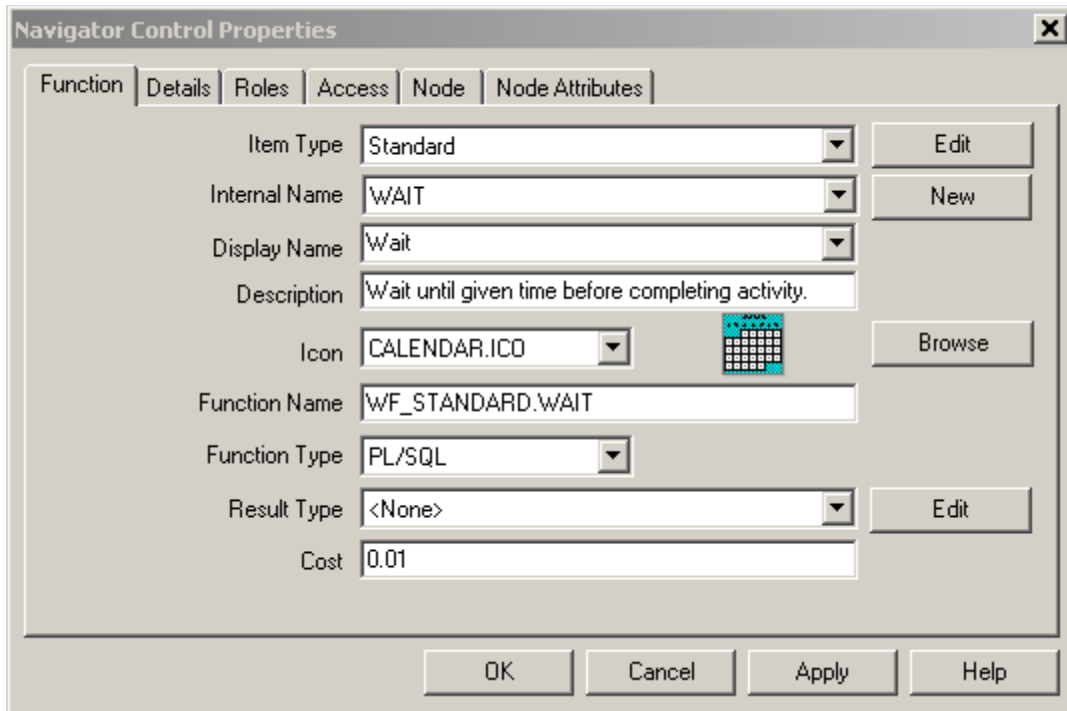
With that it is time to move to the next function to look at:

Wait

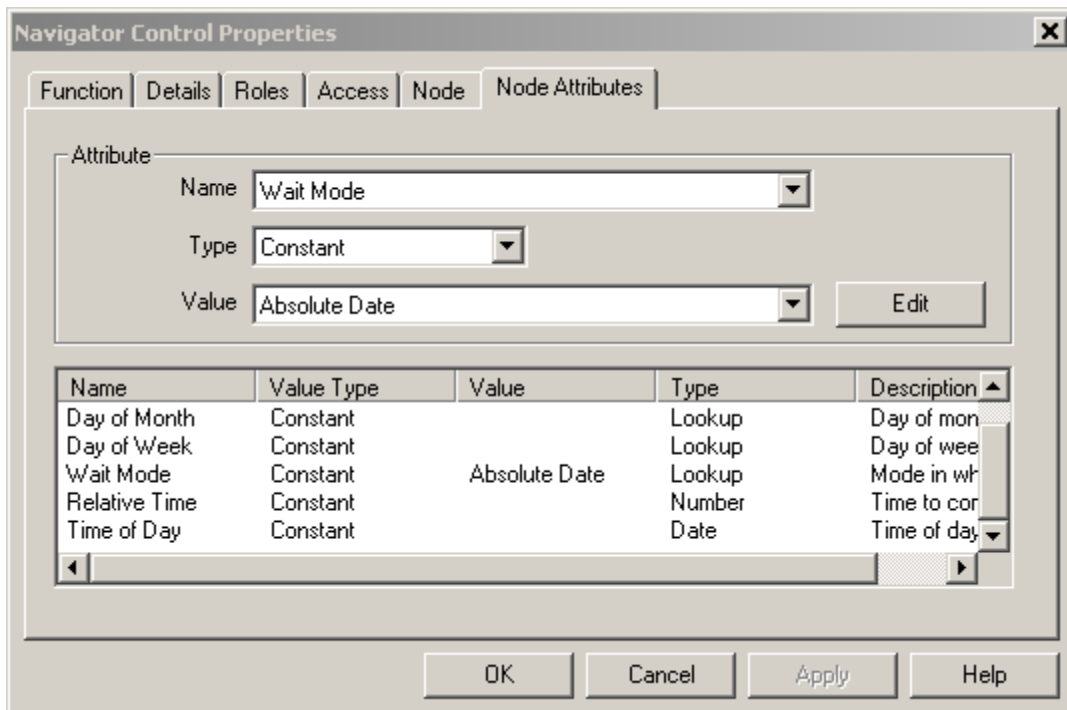
With the 'Block', we also saw the time-out. But at times, it is always required to wait a certain time, without having to wait for an external call. For these situations, the function 'Wait' is available from the 'STANDARD' itemtype.

This function takes a delay time as input parameters and then defers itself until the wait time has passed.

Let's put it in our process:



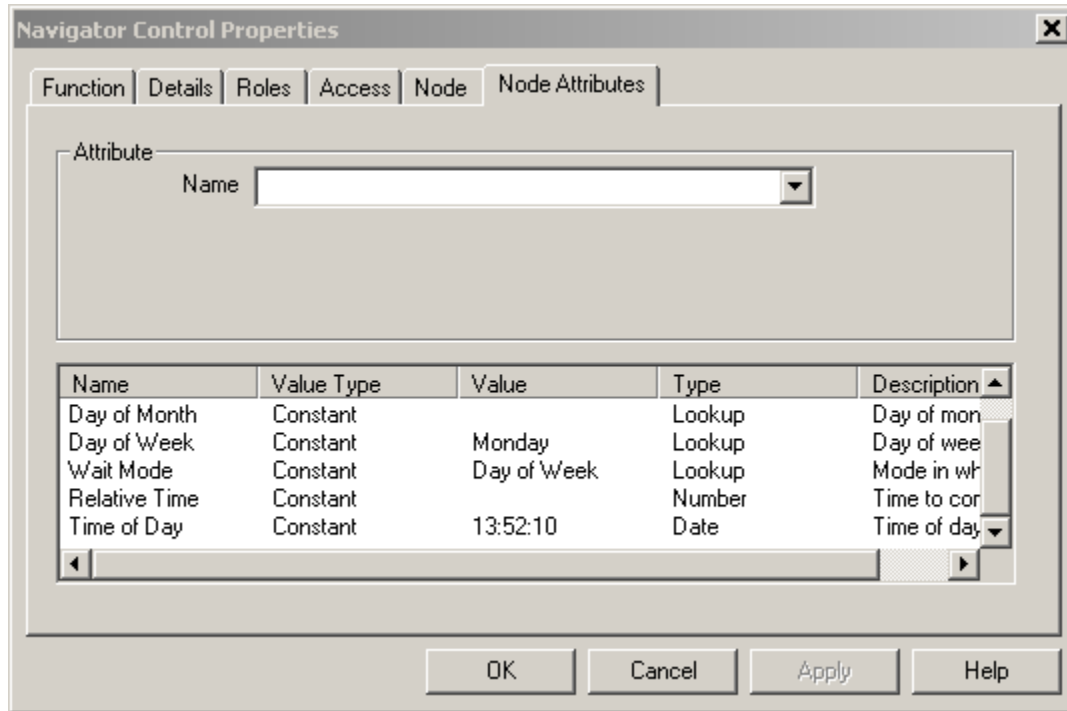
The function does not use the timeout on the 'Node' tab. But gets its input from the Node Attributes:



The first attribute to fill is the 'Wait Mode'. The options are 'Absolute Date', 'Day of Week', 'Day of Month', 'Relative Time'.

Based on the value of 'Wait Mode', you enter one of the other attributes. For 'Absolute Date', 'Day of Week' and 'Day of Month' you can also fill 'Time of Day'.

I'll first show you the 'Day of Week' mode. The same applied for 'Absolute Date' and 'Day of Month'.



These settings indicate that the process activity is to be deferred until the next Monday at 13:52:10. We can see the definition of the activity_attributes with:

```

Select Waa.Activity_name
,      Waa.Name
,      Waa.Type
,      Waa.Format
,      Waav.Text_value
,      Waav.Date_value
,      waav.number_value
From Wf_activity_attributes Waa
Join Wf_activity_attr_values Waav
on (waa.name=waav.name)
Join Wf_process_activities Wpa
On (Wpa.Instance_id=Waav.Process_activity_id
and wpa.activity_item_type=waa.activity_item_type)
Join Wf_activities Wa
on (wa.item_type=wpa.process_item_type
and Wa.Name=Wpa.Process_name
and Wa.Version=Wpa.Process_version)
Join Wf_activities Wa2
On (Wpa.Activity_item_type=Wa2.Item_type
and Wpa.Activity_name=Wa2.Name
and Wa2.Version=Waa.Activity_version
and wa2.end_date is null)
Where Wa.End_date Is Null
and Wpa.Process_item_type='DBA_TYPE'
And Wpa.Process_name='TEST_CONTROLS'
And Wpa.Activity_name='WAIT';

```

ACTIVITY_NAME	NAME	TYPE	FORMAT	TEXT_VALUE	DATE_VALUE	NUMBER_VALUE
WAIT	WAIT_ABSOLUTE_DATE	DATE				
WAIT	WAIT_DAY_OF_MONTH	LOOKUP	WFSTD_DAY_OF_MONTH			
WAIT	WAIT_DAY_OF_WEEK	LOOKUP	WFSTD_DAY_OF_WEEK	MONDAY		
WAIT	WAIT_MODE	LOOKUP	WFSTD_WAIT_MODE	DAY_OF_WEEK		
WAIT	WAIT_RELATIVE_TIME	NUMBER				
WAIT	WAIT_TIME	DATE	HH24:MI		30-12-99 13:52:10	

The query is quite complex, because you'll need to establish the process_version first. Then find the latest activity_version, to get to the activity_attribute.

Note the format on the 'WAIT_TIME'. The wait_time has a granularity of a minute. So the 10 seconds will be cut off later. The date-part of the 'WAIT_TIME' is ignored by the workflow engine. It is only there to make use of the date-type.

Now let's run the process and see the result:

KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	DUE_DATE	STATUS	RESULT	OUTBO
15	Test Workflow Controls		25-10-09 15:09:56			ACTIVE	#NULL	
15	Start	START	25-10-09 15:09:56	25-10-09 15:09:56		COMPLETE	#NULL	
15	Wait	WF_STANDARD.WAIT	26-10-09 13:52:00			DEFERRED	#NULL	8EDD9FB68A9A4B73B0C8..

Unlike the block, the item is now deferred again. There is no DUE_DATE, but an outbound_queue_id. Also note that the begin_date of the item is set to the end of the wait_time, which is the next Monday at 13:52:00 (10 seconds cut off)

Remember from the 'Defer', that it checks for a result_code to see if it should be deferred, or continued. The 'Wait' does the same thing. So to complete the item before the wait time we just re-run the process_activity:

```

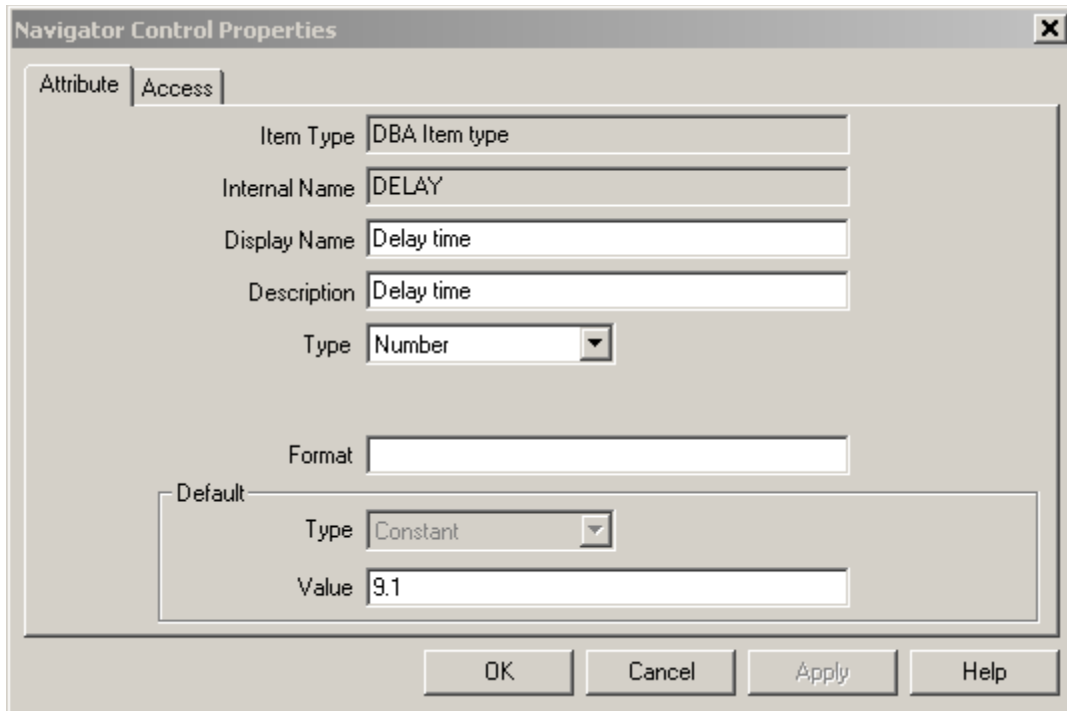
Begin
Wf_engine.Handleerror (Itemtype=> 'DBA_TYPE'
                      ,Itemkey=> '15'
                      ,Activity=> 'TEST_CONTROLS:WAIT'
                      ,Command=> 'RETRY' );
end;

```

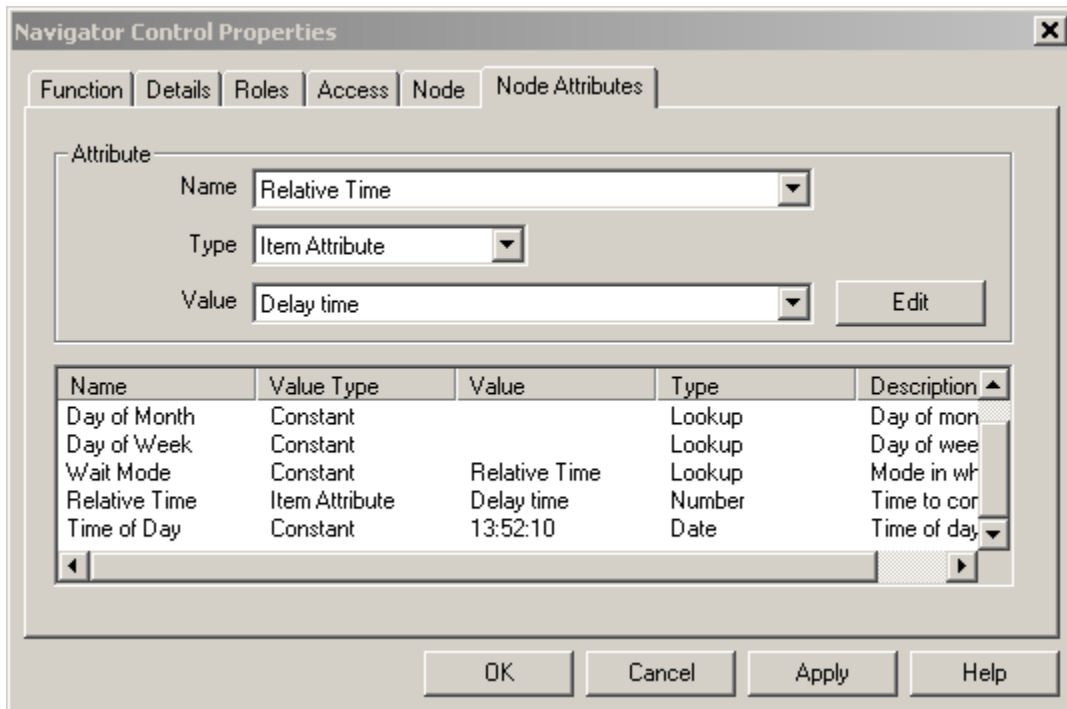
KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	DUE_DATE	STATUS	PROCESS_A	RESULT
15	Test Workflow Controls		25-10-09 15:09:56	25-10-09 15:18:40		COMPLETE	788609	#NULL
15	Start	START	25-10-09 15:09:56	25-10-09 15:09:56		COMPLETE	789307	#NULL
15	Wait	WF_STANDARD.WAIT	25-10-09 15:18:40	25-10-09 15:18:40		COMPLETE	789309	#NULL
15	End	END	25-10-09 15:18:40	25-10-09 15:18:40		COMPLETE	789305	#NULL

This completes the activity in time. Since the activity is no longer deferred, the background engine will ignore the item later on.

Now let's see the relative time. This time, we use an item attribute to set the delay:



Then we use it to set the 'Wait' event:



Now when we run the process:

```

KEY DISPLAY_NAME      FUNCTION      BEGIN_DATE    END_DATE      STATUS  PROCESS_A  RESULT  OUTBOUND_Q
16 Test Workflow Controls  START        25-10-09 15:40:28  25-10-09 15:40:28  COMPLETE  788609      #NULL
16 Start                WF_STANDARD.WAIT  03-11-09 13:52:00  DEFERRED  789617      #NULL 5ECA9EE93ED547919..

```

The 'Time of Day' has overwritten the time-part of the 'Delay Time'. If 'Time of Day' is left empty, the Delay Time would be taken as of the begin-date of the process activity.

Perhaps we'll realize now that 9 days is a bit too far away, and we want to cancel the whole item. This is done with

Abort

To abort an item, we can call 'WF_ENGINE.AbortProcess'.

```
Begin
    Wf_engine.abortprocess(itemtype=>'DBA_TYPE'
                          ,itemkey =>'16'
                          ,verify_lock => TRUE
                          ,cascade => TRUE);
End;
```

An optional parameter is 'Process', with '<process>:<activity>', in case we want to cancel a sub-process. It defaults to the root process.

And 'Result' with an optional result for this process. It defaults to '#FORCE'.

After aborting the item:

KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	STATUS	RESULT
16	Test Workflow Controls		25-10-09 15:40:28	25-10-09 16:19:57	COMPLETE	#FORCE
16	Start	START	25-10-09 15:40:28	25-10-09 15:40:28	COMPLETE	#NULL
16	Wait	WF_STANDARD.WAIT	03-11-09 13:52:00	25-10-09 16:19:57	COMPLETE	#FORCE

Of course there is also an option to only pause the item.

For this we have

Suspend and Resume

Both are in the WF_ENGINE: 'WF_ENGINE.SuspendProcess' and 'WF_ENGINE.ResumeProcess'.

Let's run the process again:

```
Begin
Wf_engine.Launchprocess(Itemtype=>'DBA_TYPE' ,Itemkey=>17,Process=>'TEST_CONTROLS');
end;
```

Once it is deferred, we can 'suspend' it:

```
Begin
Wf_engine.SuspendProcess(Itemtype=>'DBA_TYPE' ,Itemkey=> '17');
end;
```

The process itself may be deferred, but there was no impact on the 'Deferred' activity:

DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	STATUS	PROCESS_A	RESULT	OUTBOUND_QUEUE_ID
Test Workflow Controls		25-10-09 16:29:29		SUSPEND	788609	#NULL	
Start	START	25-10-09 16:29:29	25-10-09 16:29:29	COMPLETE	789615	#NULL	
Wait	WF_STANDARD.WAIT	25-10-09 13:52:00		DEFERRED	789617	#NULL	A68DA1557FAC4DC4B..

After running a background engine, the process is still suspended. But the message has been removed from the wf_deferred_table_m queue:

```
Begin
```

```
Wf_engine.Background(Itemtype=> 'DBA_TYPE' );
end;
```

DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	STATUS	PROCESS_A	RESULT	OUTBOUND_QUEUE_ID
Test Workflow Controls		25-10-09 16:54:19		SUSPEND	788609	#NULL	
Start	START	25-10-09 16:54:19	25-10-09 16:54:19	COMPLETE	789615	#NULL	
Wait	WF_STANDARD.WAIT	25-10-09 17:00:15		DEFERRED	789617	#NULL	

The 'Wait' has been re-executed, but because of the 'Suspend', the workflow engine did not continue processing. When we resume the item, the deferred activity will be re-executed, and the processing will continue immediately.

When the activity is still deferred, when the item is resumed, it will continue waiting for the delay time to pass.

Begin

```
Wf_engine.Resumeprocess(Itemtype=>'DBA_TYPE',Itemkey=>'17');
end;
```

DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	STATUS	PROCESS_A	RESULT	OUTBOUND_QUEUE_ID
Start	START	25-10-09 16:54:19	25-10-09 16:54:19	COMPLETE	789615	#NULL	
Test Workflow Controls		25-10-09 16:54:19	25-10-09 17:04:41	COMPLETE	788609	#NULL	
Wait	WF_STANDARD.WAIT	25-10-09 17:04:41	25-10-09 17:04:41	COMPLETE	789617	#NULL	
End	END	25-10-09 17:04:41	25-10-09 17:04:41	COMPLETE	789613	#NULL	

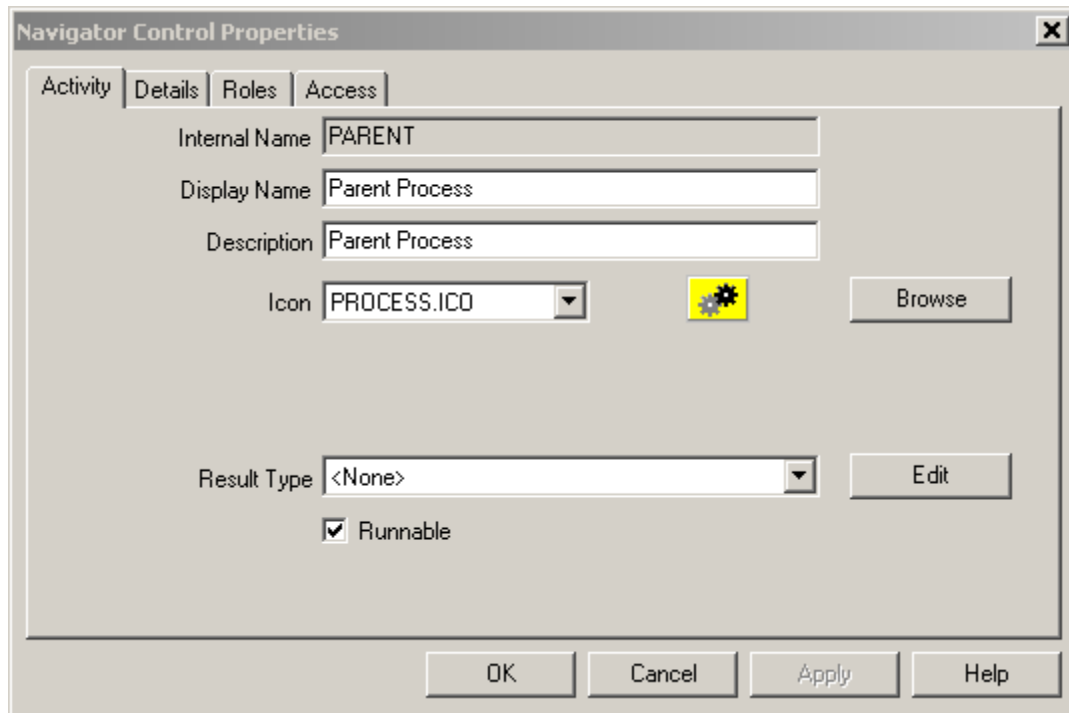
So far all the 'controls' that we've seen were confined to just one item. Now we'll see some that are spanning multiple items.

Wait for Flow / Continue Flow

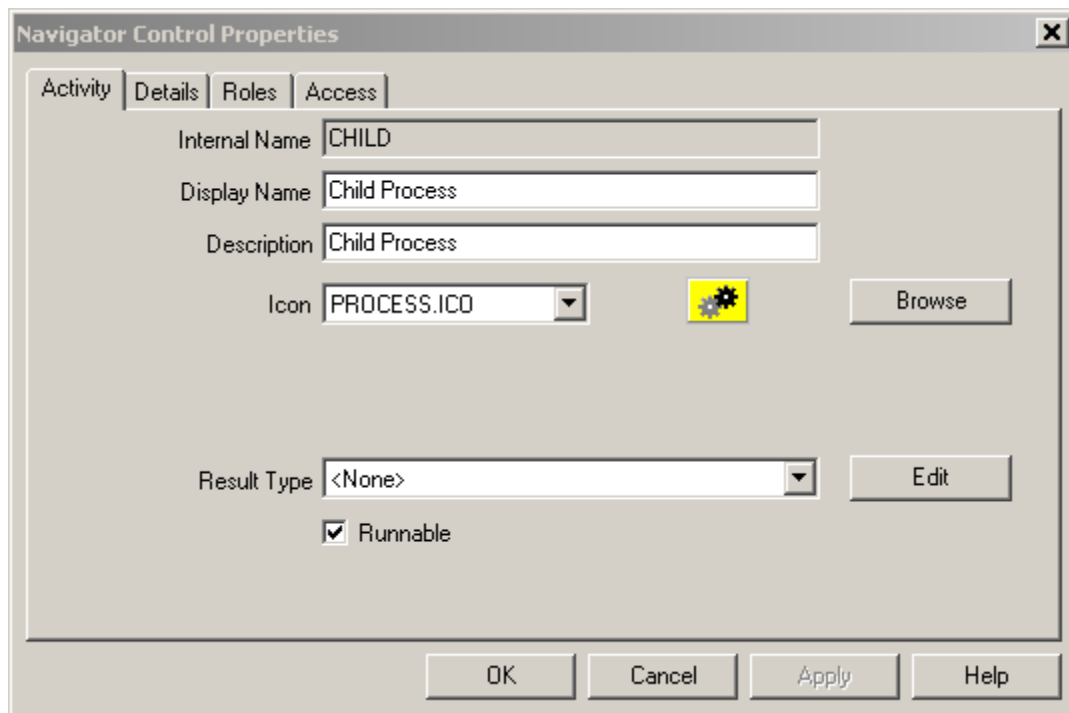
Two items can be connected together in a parent-child relationship. This may or may not affect their flow. First we are going to build 2 processes, where 1 process starts another without any relation.

Then we'll add the parent-child relation to them. And finally we'll see some examples of how parent and child can interact together.

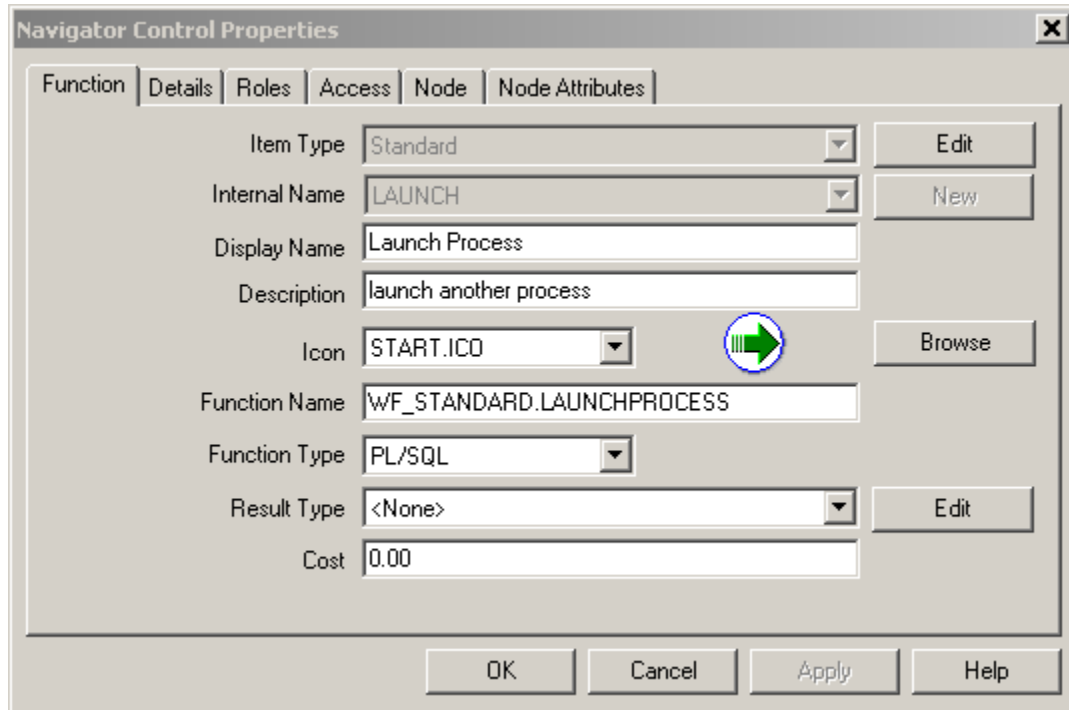
To start, we create 2 new processes. The first is 'PARENT':



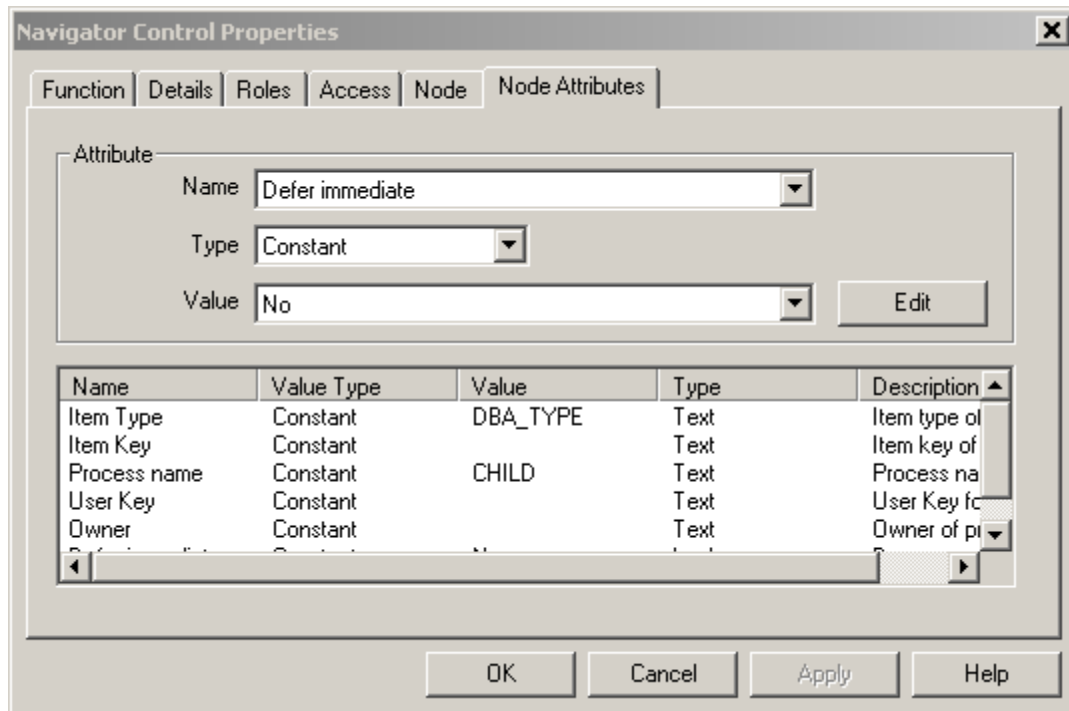
The second is 'CHILD':



On the 'PARENT', we add the function 'Launch'. Again this function is available from the 'STANDARD' itemtype. It is used to Launch (Create and Start) another process.



On the Node Attributes, we can define the process to start:



The attributes that are set here are 'Item Type' and 'Process name'. Their meaning should be obvious. Otherwise, you might want to check on [Part 1](#) of this series.

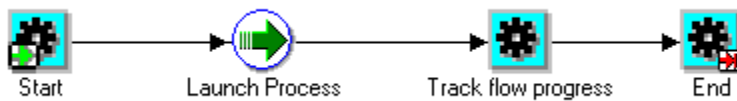
One attribute is hidden from the box. That is 'Defer Immediate'. When set to 'Y', it will defer the parent item before launching the child. This way, the launch and execution of the child will be done by a background engine.

The User Key and Owner are optional and used for future reference to the launched item.

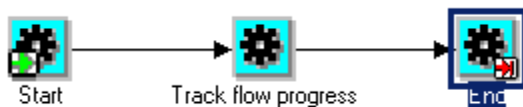
You'll have noticed that 'Item Key' is not entered. Even though we announced before that it is a mandatory value for each item. In this case, the item_key will be generated by the workflow engine as '<parent item type>:<parent item key>-<child counter>' where '<child counter>' is a counter for the number of times a child process has been launched from this parent item.

The count is kept in an automatically generated item attribute: 'LAUNCH_COUNT'. It is increased on every 'Launch' from a parent item.

Let's see how this works out. I added a 'Track Flow Progress' function on both the child and parent process. So they look like this:



And the child:



Now when we run the 'Parent', we see the following:

Begin

```

Wf_engine.Launchprocess( Itemtype=>'DBA_TYPE' , Itemkey=>1 , Process=>' PARENT' ) ;
end;
  
```

KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	STATUS	PROCESS_A	RESULT
1	Parent Process		26-10-09 23:00:26	26-10-09 23:00:26	COMPLETE	790332	#NULL
1	Start	START	26-10-09 23:00:26	26-10-09 23:00:26	COMPLETE	790675	#NULL
1	Launch Process	WF_STANDARD.LAUNCHPROCESS	26-10-09 23:00:26	26-10-09 23:00:26	COMPLETE	790677	COMPLETE
1	Track flow progress	XXX_WF_DBA.track_flow_progress	26-10-09 23:00:26	26-10-09 23:00:26	COMPLETE	790681	
1	End	END	26-10-09 23:00:26	26-10-09 23:00:26	COMPLETE	790679	#NULL

KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	STATUS	RESULT
DBA_TYPE:1-1	Child Process		26-10-09 23:00:26	26-10-09 23:00:26	COMPLETE	#NULL
DBA_TYPE:1-1	Start	START	26-10-09 23:00:26	26-10-09 23:00:26	COMPLETE	#NULL
DBA_TYPE:1-1	Track flow progress	XXX_WF_DBA.track_flow_progress	26-10-09 23:00:26	26-10-09 23:00:26	COMPLETE	
DBA_TYPE:1-1	End	END	26-10-09 23:00:26	26-10-09 23:00:26	COMPLETE	#NULL

The items completed within a second. But the parent item (actually the 'Launch Process') waited for the child item to complete. A wait or defer would have caused the child item to return control back to the parent, which would have finished then.

Now we want to establish the parent-child relationship between the items. It is not possible to set up a relationship on the process level. It needs to be established for individual items. Also, it cannot be set with the 'LaunchProcess' function. It can be done from the parent between a 'CreateProcess' and a 'StartProcess'. Or you can establish it from the child workflow. We will use the last option.

For the child to set the parent-child relation, it needs to be aware of its parent. So we use a set of PL/SQL variables to communicate between the items.

We add a new function to our package 'XXX_WF_DBA'.

Package definition:

```
g_parent_item_type Varchar2(200);  
g_parent_item_key varchar2(200);
```

```
PROCEDURE init_parent(p_item_type IN VARCHAR2  
                     ,p_item_key IN VARCHAR2  
                     ,p_actid IN NUMBER  
                     ,P_funcmode In Varchar2  
                     ,P_result Out Varchar2);
```

Package body:

```
PROCEDURE init_parent(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  
    Xxx_wf_dba.G_parent_item_type := P_item_type;  
    Xxx_wf_dba.G_parent_item_key := p_item_key;  
  End If;  
END;
```

And of course an extra function in our 'Parent Process':

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

- Item Type: DBA Item type
- Internal Name: INIT_PARENT
- Display Name: Initialize Parent Flow
- Description: Initialize Parent Flow
- Icon: FUNCTION.ICO (with a gear icon)
- Function Name: XXX_WF_DBA.init_parent
- Function Type: PL/SQL
- Result Type: <None>
- Cost: 0.00

In the child process, we use our 'init' procedure. (XXXAS: See part x) For this I expanded it to assign the parent.

```

PROCEDURE init(
    p_item_type IN VARCHAR2 ,
    p_item_key  IN VARCHAR2 ,
    p_actid    IN NUMBER ,
    p_funcmode IN VARCHAR2 ,
    p_result OUT VARCHAR2)
IS
    v_status  Varchar2(8);
    v_result  varchar2(30);
BEGIN
    IF p_funcmode='RUN' THEN
        wf_engine.SetItemAttrText(itemtype=>p_item_type
                                ,itemkey =>p_item_key
                                ,Aname=>'INSTANCE_NAME'
                                ,Avalue=>Sys_context('USERENV','DB_NAME'));
        Wf_engine.Setitemparent(Itemtype=>p_item_type
                                ,Itemkey=>p_item_key
                                ,Parent_itemtype=> Xxx_wf_dba.G_parent_item_type
                                ,Parent_itemkey=> Xxx_wf_dba.G_parent_item_key
                                ,parent_context => NULL);
    End If;
    Wf_item_activity_status.Result(p_item_type, p_item_key, p_actid, v_status, v_result);
    If (v_result = Wf_engine.Eng_null) Then
        P_result := Wf_engine.Eng_completed||':'||Wf_engine.Eng_null;
    Else
        P_result := Wf_engine.Eng_deferred;
    End If;
End;

```

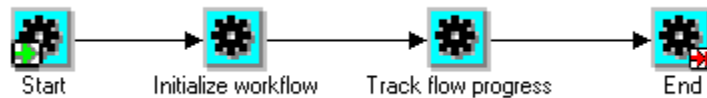
This sets the parent flow to the itemtype/itemkey defined in the variables. The 'parent_context' parameter is optional. You should set this when you start multiple child processes from your parent. In that case it will be used to identify the different children.

Take note of the 'p_result'. I changed this to 'deferred'. That way, we can see the completion of the parent while the child is still active.

In the end this is the parent:



And the child:



After running the parent process we get this result:

KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	DUE_DATE	STATUS	RESULT
2	Parent Process		31-10-09 17:22:32	31-10-09 17:22:34	31-10-09 17:22:34	COMPLETE	#NULL
2	Start	START	31-10-09 17:22:33	31-10-09 17:22:33	31-10-09 17:22:33	COMPLETE	#NULL
2	Initialize Parent Flow	XXX_WF_DBA.init_parent	31-10-09 17:22:33	31-10-09 17:22:33	31-10-09 17:22:33	COMPLETE	#NULL
2	Launch Process	WF_STANDARD.LAUNCHPROCESS	31-10-09 17:22:33	31-10-09 17:22:34	31-10-09 17:22:34	COMPLETE	COMPLETE
2	Track flow progress	XXX_WF_DBA.track_flow_progress	31-10-09 17:22:34	31-10-09 17:22:34	31-10-09 17:22:34	COMPLETE	COMPLETE
2	End	END	31-10-09 17:22:34	31-10-09 17:22:34	31-10-09 17:22:34	COMPLETE	#NULL

And the child process:

KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	STATUS	RESULT
DBA_TYPE:2-1	Child Process		31-10-09 17:22:33		ACTIVE	#NULL
DBA_TYPE:2-1	Start	START	31-10-09 17:22:33	31-10-09 17:22:33	COMPLETE	#NULL
DBA_TYPE:2-1	Initialize workflow	XXX_WF_DBA.init	31-10-09 17:22:33		DEFERRED	#NULL

Nicely deferred at the initialize. The interesting thing is at the 'WF_ITEMS' table though:

```

Select Item_type
, Item_key
, Root_activity
, Parent_item_type
, Parent_item_key
, Parent_context
, Begin_date
, end_date
From Wf_items
where item_type='DBA_TYPE'
and item_key in ('2','DBA_TYPE:2-1');
  
```

ITEM_TYPE	ITEM_KEY	ROOT_ACTIVITY	PARENT_ITEM_TYPE	PARENT_ITEM_KEY	PARENT_C	BEGIN_DATE	END_DATE
DBA_TYPE	2	PARENT				31-10-09 17:22:31	31-10-09 17:22:34
DBA_TYPE	DBA_TYPE:2-1	CHILD	DBA_TYPE	2		31-10-09 17:22:33	

Item '2' is now the parent of 'DBA_TYPE:2-1' . And the parent completed after the child was deferred.

So far so good. Let's take it one step further. We'll defer the parent item until the child item has completed. For this we add a 'Wait for Flow'to the parent :

Navigator Control Properties

Function | Details | Roles | Access | Node | Node Attributes

Item Type: Standard [Edit]

Internal Name: WAITFORFLOW [New]

Display Name: Wait For Flow

Description: Wait for flow to complete

Icon: STOP.ICO [Browse]

Function Name: WF_STANDARD.WAITFORFLOW

Function Type: PL/SQL

Result Type: <None> [Edit]

Cost: 0.01

OK Cancel Apply Help

The function has 2 node attributes:

Navigator Control Properties

Function | Details | Roles | Access | Node | Node Attributes

Attribute

Name []

Name	Value Type	Value	Type	Description
Continuation Acti...	Constant	CONTINUEFLOW	Text	Label of activit
Continuation Flow	Constant	Detail	Lookup	Location of coi

OK Cancel Apply Help

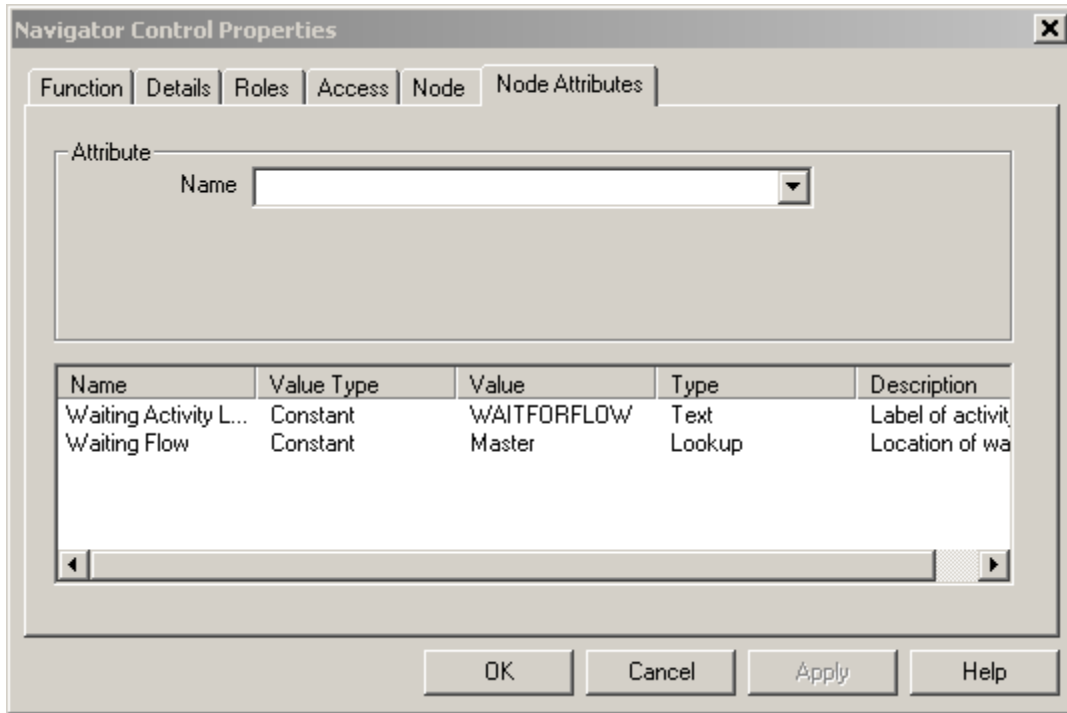
The first attribute 'Continuation Activity Label' refers to the activity label of its counterpart on the child workflow. The second attribute 'Continuation Flow' indicates whether the 'Continue Flow' activity is in the parent (Master) or child (Detail) item. In our case the 'Wait for Flow' is on the parent, so the 'ContinueFlow' is in the 'Detail'. Together these attributes uniquely identify a 'Continue Flow' activity. The parent process will look like this.



On the child process we add a 'Continue Flow':

Function	Details	Roles	Access	Node	Node Attributes
Item Type	Standard				
Internal Name	CONTINUEFLOW				
Display Name	Continue Flow				
Description	Continue flow				
Icon	START.ICO				
Function Name	WF_STANDARD.CONTINUEFLOW				
Function Type	PL/SQL				
Result Type	<None>				
Cost	0.01				

It also has 2 node attributes:



As you can see, these attributes refer to the 'Wait for Flow' activity on the parent.

The whole process looks like this:



And when we run our parent process we see the following result:

KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	STATUS	RESULT
3	Parent Process		01-11-09 12:03:18		ACTIVE	#NULL
3	Start	START	01-11-09 12:03:18	01-11-09 12:03:18	COMPLETE	#NULL
3	Initialize Parent Flow	XXX_WF_DBA.init_parent	01-11-09 12:03:18	01-11-09 12:03:18	COMPLETE	#NULL
3	Launch Process	WF_STANDARD.LAUNCHPROCESS	01-11-09 12:03:18	01-11-09 12:03:18	COMPLETE	COMPLETE
DBA_TYPE:3-1	Child Process		01-11-09 12:03:18		ACTIVE	#NULL
DBA_TYPE:3-1	Start	START	01-11-09 12:03:18	01-11-09 12:03:18	COMPLETE	#NULL
DBA_TYPE:3-1	Initialize workflow	XXX_WF_DBA.init	01-11-09 12:03:18		DEFERRED	#NULL
3	Track flow progress	XXX_WF_DBA.track_flow_progress	01-11-09 12:03:18	01-11-09 12:03:18	COMPLETE	#NULL
3	Wait For Flow	WF_STANDARD.WAITFORFLOW	01-11-09 12:03:18		NOTIFIED	

I selected both '3' and 'DBA_TYPE:3-1', so we can see the order in which they are executed.

The parent item is now waiting for the 'Wait For Flow'. The child is still deferred on the 'Initialize workflow'. Now let's run a background engine:

KEY	DISPLAY_NAME	FUNCTION	BEGIN_DATE	END_DATE	STATUS	RESULT
3	Parent Process		01-11-09 12:03:18	01-11-09 12:10:42	COMPLETE	#NULL
3	Start	START	01-11-09 12:03:18	01-11-09 12:03:18	COMPLETE	#NULL
3	Initialize Parent Flow	XXX_WF_DBA.init_parent	01-11-09 12:03:18	01-11-09 12:03:18	COMPLETE	

3	Launch Process	WF_STANDARD.LAUNCHPROCESS	01-11-09 12:03:18	01-11-09 12:03:18	COMPLETE	COMPLETE
DBA_TYPE:3-1	Child Process		01-11-09 12:03:18	01-11-09 12:10:42	COMPLETE	#NULL
DBA_TYPE:3-1	Start	START	01-11-09 12:03:18	01-11-09 12:03:18	COMPLETE	#NULL
3	Track flow progress	XXX_WF_DBA.track_flow_progress	01-11-09 12:03:18	01-11-09 12:03:18	COMPLETE	
3	Wait For Flow	WF_STANDARD.WAITFORFLOW	01-11-09 12:03:18	01-11-09 12:10:42	COMPLETE	#NULL
DBA_TYPE:3-1	Initialize workflow	XXX_WF_DBA.init	01-11-09 12:10:42	01-11-09 12:10:42	COMPLETE	#NULL
DBA_TYPE:3-1	Track flow progress	XXX_WF_DBA.track_flow_progress	01-11-09 12:10:42	01-11-09 12:10:42	COMPLETE	
DBA_TYPE:3-1	Continue Flow	WF_STANDARD.CONTINUEFLOW	01-11-09 12:10:42	01-11-09 12:10:42	COMPLETE	#NULL
3	End	END	01-11-09 12:10:42	01-11-09 12:10:42	COMPLETE	#NULL
DBA_TYPE:3-1	End	END	01-11-09 12:10:42	01-11-09 12:10:42	COMPLETE	#NULL

As you can see, the 'Continue Flow' completed the 'Wait for Flow', and the remaining activities in item '3'. Then it returned to complete item 'DBA_TYPE:3-1'.

The 'Wait for Flow', 'Continue Flow' activities are very stable. However, sometimes it seems as if the 'Continue Flow' didn't work. In these cases, the first step is to verify that both the 'Continue Flow' and the 'Wait for Flow' were able to identify the correct counterpart. There has to be a notified 'Wait for Flow' with the correct activity label.

Also the 'Wait for Flow' will wait for ALL child items, when more than one is started. So be careful to look for all child items when the parent is waiting.

With that we wrap up part 4 of this series. The Workflow Directory Service will be in part 5.