

Moving Around a Process in Oracle Workflow

Author: Matthew Searle

Document Date: 30th April 2007

Copyright 2007 TS Fifteen Ltd.

Introduction

When using Oracle Workflow, the general rule of thumb is that transitions between activities are managed through transitioning from one activity to the next in the diagram. However, there are certain circumstances where there may be a requirement to jump from one activity to another, which is not directly linked in the process diagram. For example, if there is a long-running process, and the business logic has been changed since the process began, it may be that the business requires all the running processes to skip a number of activities in the process. Unless the process is aborted and restarted, there is no easy mechanism to do this programmatically.

Scenarios for Breaking the Process Flow

There are a few scenarios that are immediately obvious where the ability to jump between activities becomes desirable:

- The workflow process needs to bypass a number of activities in certain circumstances, but diagramming the process would produce an overly complex diagram, or would result in confusion in the diagram.
- Long-running processes are being migrated from one workflow definition to another (either from one Oracle Workflow definition to a newer one, or from a third-party solution into Oracle Workflow). In this scenario, the business cannot allow a long-running process to be restarted, so there needs to be a mechanism to enable the process to start from a particular point in the process.

How it Works

The procedure that is used to “jump” between activities has the following signature:

```
PROCEDURE wf_jump ( p_itemkey          IN VARCHAR2
                   , p_activity_id     IN PLS_INTEGER  DEFAULT NULL
                   , p_process_name     IN VARCHAR2    DEFAULT NULL
                   , p_instance_label   IN VARCHAR2    DEFAULT NULL
                   , p_itemtype         IN VARCHAR2    DEFAULT NULL
                   , p_process_date     IN DATE        DEFAULT NULL )
```

The only manual parameter is the item key for the process that you are manipulating. If you know the activity ID for the target activity to jump to, then you should supply it in the second parameter. Since this is a system generated value, the process name, instance label and item type can be provided instead - the procedure will then derive the activity ID for the activity. Including the process name is vital, since the internal name of the activity may not be unique within the item type - for example jumping to the START activity in a sub-process. Using the instance label instead of the activity name enables

the procedure to jump within a process where there are multiple instances of the same activity, for example a process that uses a number of standard “Assign” activities. The final parameter should be used if the activity ID is not known, but later versions of the process have been deployed to the database. In this scenario, you need to supply the date of the process (or any date when that version of the process was active), so that the procedure can correctly identify which version of the activity to reference.

Once the procedure has determined what the activity ID for the target activity is (either by having the value passed into the code, or deriving it from the other parameters passed in), the core details of the item instance are retrieved - the item type, process name, instance label and process version are retrieved from the `WF_PROCESS_ACTIVITIES` table.

The procedure then determines the activity ID and notification ID for the activity which is currently waiting completion. The record for this activity will be updated to indicate that a manual transition has taken place, and that the process has jumped from this activity to a different one. This is performed as a direct table update on the record in the `WF_ITEM_ACTIVITY_STATUSES` table, rather than using an API call to update the record.

If there are any sub-processes which are waiting to be completed, and the process is jumping out of them into the main process, then these will need to be marked as being complete as well. Again, this is performed as a direct table update rather than using an API.

If the activity which was `NOTIFIED` was a notification, then the notification is closed, so that it no longer appears in a Worklist. This is yet another direct table update on `WF_NOTIFICATIONS`.

The final stage of the procedure is to invoke the native Workflow error handling API to get the Workflow engine to “retry” the activity that has been transitioned to. By using the standard API, this ensures that the Workflow engine executes the activity and moves the process on as necessary.

Examples

I’ve built a very straightforward Workflow to show how the code can be used to jump between activities. The Workflow comprises one main process, three sub-processes and three notifications which all require a response. The notification recipient has been hard-coded as `SYSADMIN`, for ease of testing. Obviously, this is just an easy example - the real-world applications work with a Workflow of any complexity.

The following figures show the process diagrams:

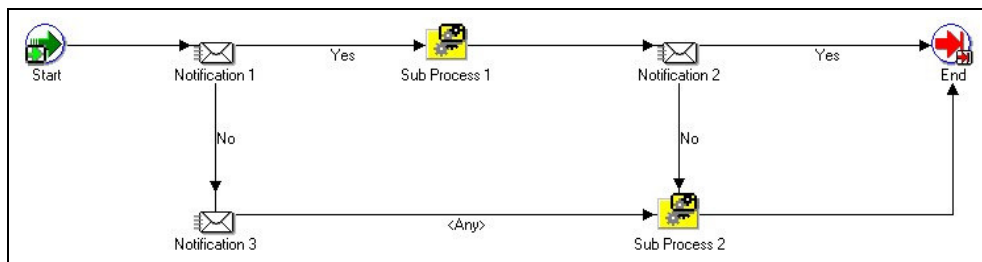


Figure 1 - Main Process

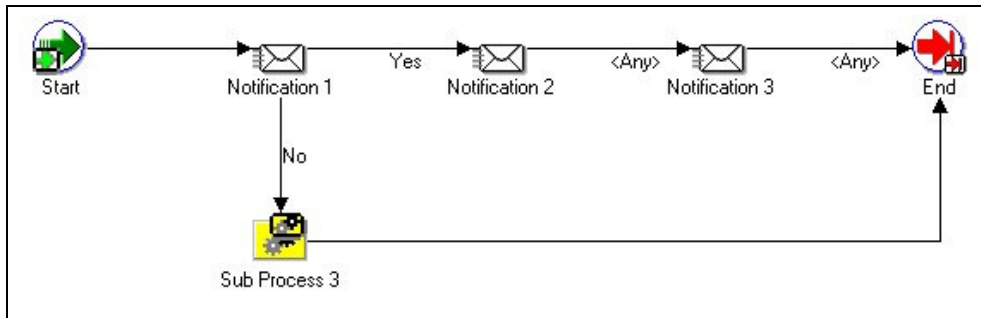


Figure 2 - Sub Process 1

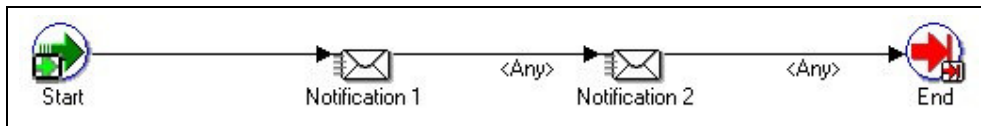


Figure 3 - Sub Process 2

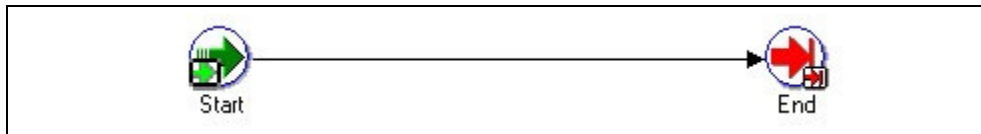


Figure 4 - Sub Process 3

With the workflow deployed to the database, the following query will detail the different activities within the item type:

```

SELECT wpa.process_name
,      wpa.activity_name
,      wpa.instance_id      actid
,      wpa.instance_label
FROM   wf_process_activities wpa
,      wf_activities        wfa
WHERE  wfa.item_type        = wpa.process_item_type
AND    wfa.name             = wpa.process_name
AND    wfa.version          = wpa.process_version
AND    wpa.process_item_type = 'JUMPING'
AND    wpa.process_name     != 'ROOT'
ORDER BY wpa.process_name, wpa.activity_name;

```

The query (in my instance) returns the following records:

PROCESS_NAME	Activity Name	ACTID	INSTANCE_LABEL
MAIN	END	572881	END
MAIN	NTF1	572887	NTF1
MAIN	NTF2	572893	NTF2
MAIN	NTF3	572891	NTF3
MAIN	START	572883	START
MAIN	SUBPROC1	572885	SUBPROC1
MAIN	SUBPROC2	572889	SUBPROC2
SUBPROC1	END	572899	END
SUBPROC1	NTF1	572903	NTF1
SUBPROC1	NTF2	572905	NTF2

SUBPROC1	NTF3	572907	NIF3
SUBPROC1	START	572901	START
SUBPROC1	SUBPROC3	572909	SUBPROC3
SUBPROC2	END	572910	END
SUBPROC2	NTF1	572914	NIF1
SUBPROC2	NTF2	572916	NIF2
SUBPROC2	START	572912	START
SUBPROC3	END	572918	END
SUBPROC3	START	572920	START

The first step is to launch a new process, which will send notification 1 to SYSADMIN, and wait for a response:

```
SQL> exec wf_engine.launchprocess('JUMPING','1','MAIN');
PL/SQL procedure successfully completed.

SQL> commit;
Commit complete.
```

Here's the process status at the moment:

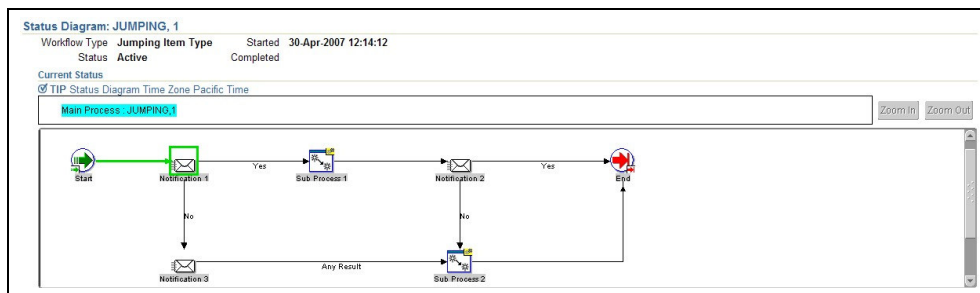


Figure 5 - Process launched by the code above

Example 1 - Jump to Sub Process 2 using Activity ID

In this example, we'll use the activity ID that we retrieved for the second sub-process (572889), to jump straight to that node.

```
SQL> BEGIN
2   wf_jump ( p_itemkey      => '1'
3             , p_activity_id => 572889
4             , p_process_name => NULL
5             , p_instance_label => NULL
6             , p_itemtype    => NULL
7             , p_process_date => NULL );
8   commit;
9   END;
10  /

PL/SQL procedure successfully completed.
```

The process jumps straight to Sub Process 2, as shown in Figure 6.

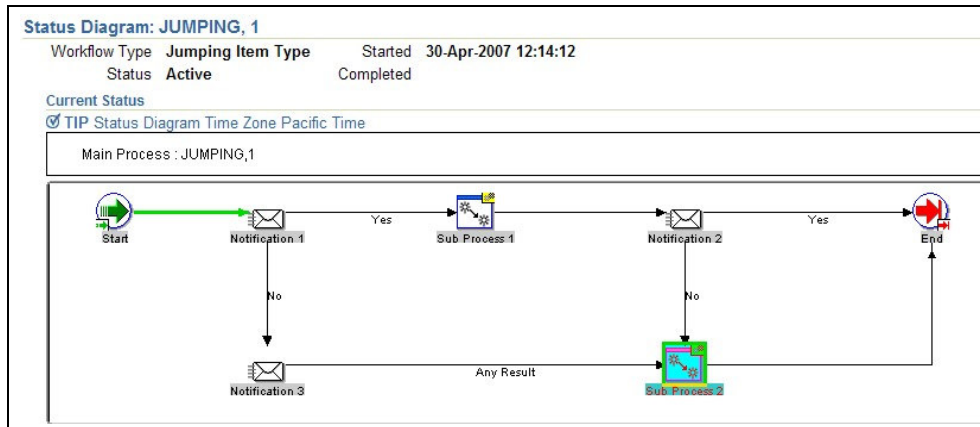


Figure 6 - Jumping to Sub Process 2

An extract from running the WFSTAT script on the process shows that the first notification has a result of “Manual Transition” - this is populated by the jumping code to show that the notification was not responded to in typical fashion.

Activity	Status	RESULT	User	NID	Status
ROOT/MAIN	ACTIVE	#NULL			
MAIN/START	COMPLETE	#NULL			
MAIN/NTF1	COMPLETE	Manual Transition	SYSADMIN	838511	CLOSED
MAIN/SUBPROC2	ACTIVE	#NULL			
SUBPROC2/START	COMPLETE	#NULL			
SUBPROC2/NTF1	NOTIFIED		SYSADMIN	838512	OPEN

Example 2 - Jump to Notification 2 in Sub Process 1 using Instance Label

In this second example, rather than use the internal activity ID for the target activity, we’ll pass in the name of the process and the instance label for the node. This method of jumping between activities can be used (for example) from within other code, without the need to reference an ID over which you have no control. If you were implementing this logic in your system, then this is the method that you should use, since it allows for easier development and migration between environments.

```
SQL> BEGIN
  2   wf_jump ( p_itemkey      => '2'
  3             , p_activity_id => NULL
  4             , p_process_name => 'SUBPROC1'
  5             , p_instance_label => 'NTF2'
  6             , p_itemtype     => 'JUMPING'
  7             , p_process_date  => NULL );
  8   commit;
  9   END;
 10  /
```

PL/SQL procedure successfully completed.

Once the code has been executed, the process monitor shows that the process has jumped from the starting point (Figure 7) to the Sub-Process (Figure 8), straight to the notification (Figure 9).

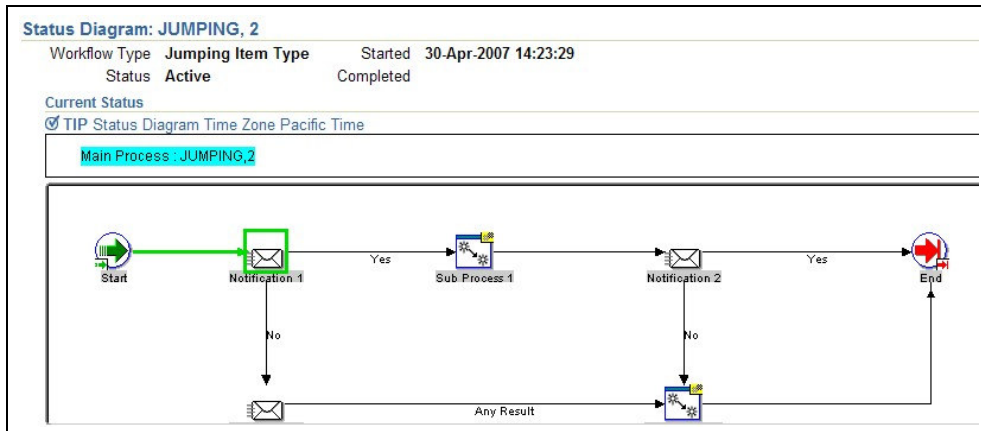


Figure 7 - New Process (Item Key 2) Waiting to Jump

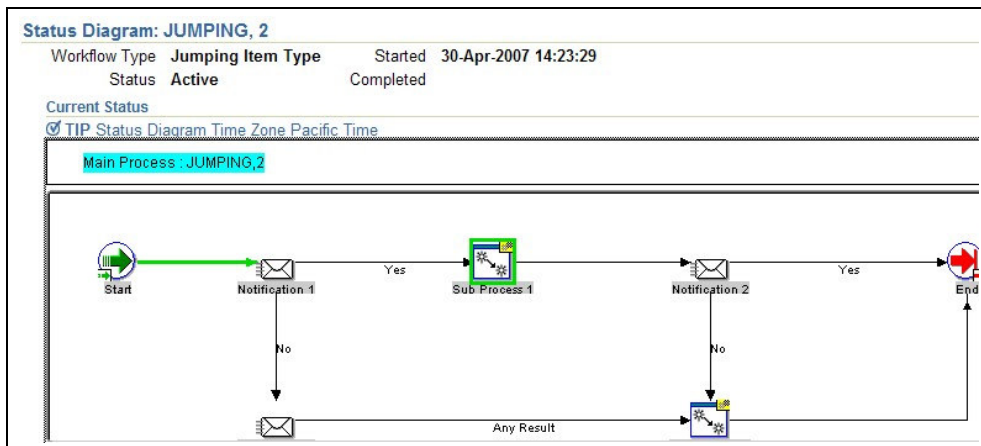


Figure 8 - Jump straight to Sub Process 1

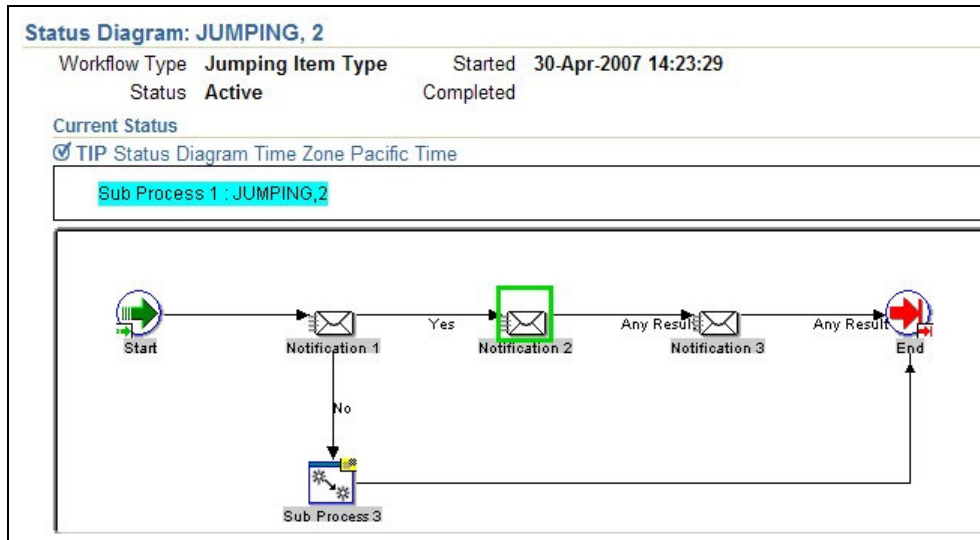


Figure 9 - Jump to Notification 2

Example 3 - Jump to Different Activities by Instance Label

In this final example, I have created a new process which includes only “Notification 1” a number of times. Figure 10 shows the diagram, showing the instance labels for each activity.

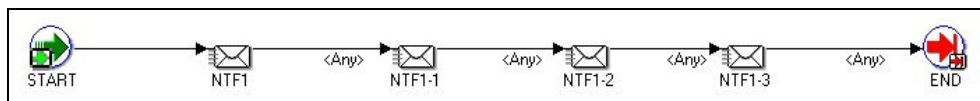


Figure 10 - Process 4

Having launched a new instance of the process (item key 3), the process is blocked at the first notification, awaiting a response. The code below will jump the process from the first notification to the last, using the instance label. The process diagrams are shown in figures 11 and 12.

```
SQL> BEGIN
2   wf_jump ( p_itemkey      => '3'
3             , p_activity_id => NULL
4             , p_process_name => 'PROCESS4'
5             , p_instance_label => 'NTF1-3'
6             , p_itemtype     => 'JUMPING'
7             , p_process_date  => NULL );
8   commit;
9   END;
10  /
```

PL/SQL procedure successfully completed.

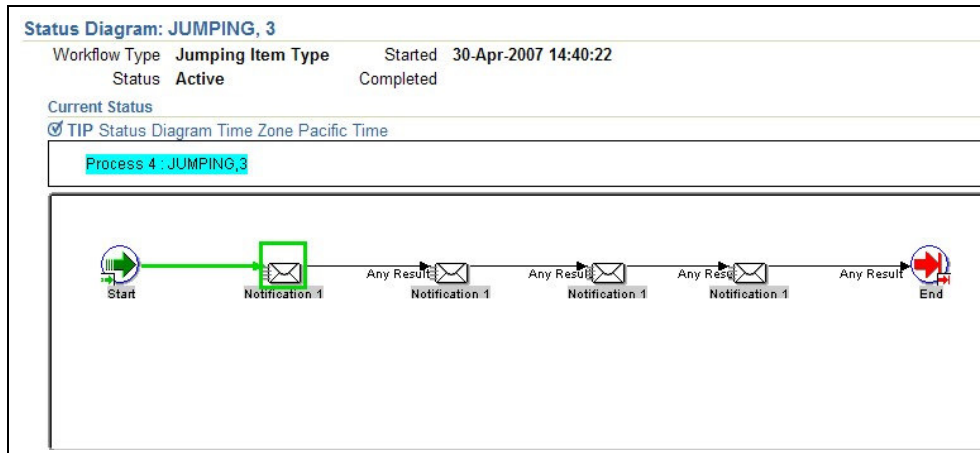


Figure 11 - Initial Process

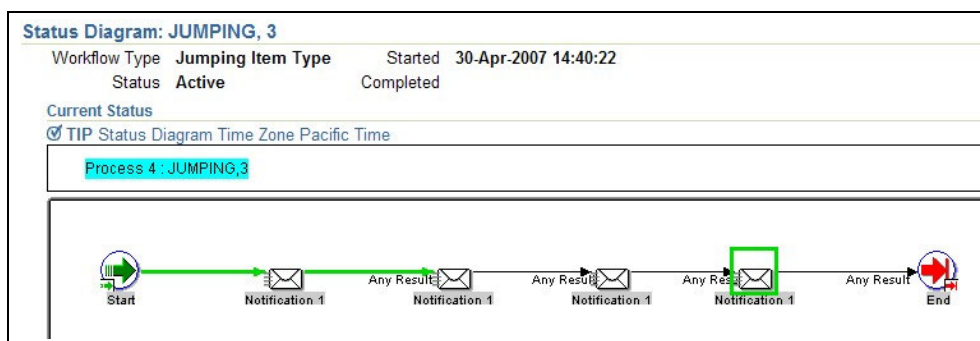


Figure 12 - Process after Jumping to Instance NTF-3

Conclusion

The code provided offers an easy mechanism for developers or system administrators to jump from one point in the process to another (possibly unrelated) point in the process. Until release 2.6.3 of Oracle Workflow, this was possible for workflow administrators to do, using the “Expedite” functionality. This was removed in release 2.6.3 as it contains a potential process security flaw, in that the process could be expedited from any particular point. This code reintroduces this functionality, and as such, should be used with great care.

Invitation for Comments

If you have any comments or criticisms of this white paper, the code supplied, or topics covered, please contact us at whitepapers@workflowfaq.com.

If you have any topics that you would like to see covered in more detail, please send your suggestions to the same email address. Whilst we welcome your comments or suggestions, we are unable to respond to each email individually.

Code

```
PROCEDURE wf_jump ( p_itemkey      IN VARCHAR2
                  , p_activity_id  IN PLS_INTEGER  DEFAULT NULL
                  , p_process_name  IN VARCHAR2    DEFAULT NULL
                  , p_instance_label IN VARCHAR2    DEFAULT NULL
                  , p_itemtype      IN VARCHAR2    DEFAULT NULL
                  , p_process_date  IN DATE        DEFAULT NULL ) IS

CURSOR c_get_actid ( cp_itemtype      IN VARCHAR2
                   , cp_instance_label IN VARCHAR2
                   , cp_process_name   IN VARCHAR2 ) IS
SELECT instance_id
FROM   wf_process_activities
WHERE  process_item_type = cp_itemtype
AND    instance_label    = cp_instance_label
AND    process_name      = cp_process_name
ORDER BY process_version DESC;

CURSOR c_get_actid_date ( cp_itemtype      IN VARCHAR2
                        , cp_instance_label IN VARCHAR2
                        , cp_process_name   IN VARCHAR2
                        , cp_process_date   IN DATE ) IS

SELECT wpa.instance_id
FROM   wf_process_activities  wpa
,      wf_activities          wfa
WHERE  wfa.item_type          = wpa.process_item_type
AND    wfa.name               = wpa.process_name
AND    wfa.version            = wpa.process_version
AND    wpa.process_item_type  = cp_itemtype
AND    wpa.instance_label     = cp_instance_label
AND    wpa.process_name       = cp_process_name
AND    wfa.begin_date         < cp_process_date
AND    NVL(wfa.end_date,SYSDATE) > cp_process_date;

CURSOR c_get_item_details ( cp_actid IN PLS_INTEGER ) IS
SELECT process_item_type  itemtype
,      process_name
,      instance_label
,      process_version
FROM   wf_process_activities
WHERE  instance_id = cp_actid;

CURSOR c_current_details ( cp_itemtype IN VARCHAR2
                          , cp_itemkey IN VARCHAR2 ) IS
SELECT process_activity  actid
,      notification_id  nid
FROM   wf_item_activity_statuses  wias
,      wf_process_activities      wpa
WHERE  wias.process_activity = wpa.instance_id
AND    wias.item_type        = cp_itemtype
AND    wias.item_key         = cp_itemkey
AND    wias.activity_status  = 'NOTIFIED';

CURSOR c_subprocesses ( cp_itemtype      IN VARCHAR2
                      , cp_process_version IN PLS_INTEGER
```

```

, cp_actid          IN PLS_INTEGER ) IS
SELECT DISTINCT instance_id  actid
FROM   wf_process_activities wpa
WHERE  process_item_type = cp_itemtype
AND    process_version   = cp_process_version
AND    process_name      != 'ROOT'
START WITH instance_id    = cp_actid
CONNECT BY activity_name = PRIOR process_name;

r_current_details      c_current_details%ROWTYPE;
r_item_details         c_get_item_details%ROWTYPE;

e_missing_parameters   EXCEPTION;
e_missing_parameters#  PLS_INTEGER := -20000;
e_missing_actid       EXCEPTION;
e_missing_actid#      PLS_INTEGER := -20001;

PRAGMA EXCEPTION_INIT ( e_missing_parameters,      -20000 );
PRAGMA EXCEPTION_INIT ( e_missing_actid,          -20001 );

v_actid                PLS_INTEGER;
v_process_version      PLS_INTEGER;

BEGIN
  IF p_activity_id IS NULL THEN

    -- If activity ID is not passed in, then need to derive the activity ID
    -- from the workflow tables
    IF p_instance_label IS NULL
    OR p_itemtype IS NULL
    OR p_process_name IS NULL THEN
      -- Missing mandatory parameters
      RAISE e_missing_parameters;
    END IF;

    IF p_process_date IS NOT NULL THEN
      OPEN c_get_actid_date ( cp_itemtype      => p_itemtype
                             , cp_instance_label => p_instance_label
                             , cp_process_name  => p_process_name
                             , cp_process_date  => p_process_date );
      FETCH c_get_actid_date INTO v_actid;
      CLOSE c_get_actid_date;
    ELSE
      OPEN c_get_actid ( cp_itemtype      => p_itemtype
                        , cp_instance_label => p_instance_label
                        , cp_process_name  => p_process_name );
      FETCH c_get_actid INTO v_actid;
      CLOSE c_get_actid;
    END IF;

    IF v_actid IS NULL THEN
      RAISE e_missing_actid;
    END IF;
  ELSE
    v_actid := p_activity_id;
  END IF;

```

```

OPEN c_get_item_details ( cp_actid => v_actid );
FETCH c_get_item_details INTO r_item_details;
CLOSE c_get_item_details;

OPEN c_current_details ( cp_itemtype => r_item_details.itemtype
                        , cp_itemkey => p_itemkey );
FETCH c_current_details INTO r_current_details;
CLOSE c_current_details;

UPDATE wf_item_activity_statuses
SET    activity_status      = 'COMPLETE'
,      activity_result_code = 'Manual Transition'
,      end_date             = SYSDATE
WHERE  item_type           = r_item_details.itemtype
AND    item_key            = p_itemkey
AND    process_activity    = r_current_details.actid;

FOR i IN c_subprocesses ( cp_itemtype      => r_item_details.itemtype
                        , cp_process_version => r_item_details.process_version
                        , cp_actid         => r_current_details.actid ) LOOP
    UPDATE wf_item_activity_statuses
    SET    activity_status = 'COMPLETE'
    ,      activity_result_code = 'Manual Transition'
    ,      end_date         = SYSDATE
    WHERE  item_key        = p_itemkey
    AND    process_activity = i.actid
    AND    item_type       = r_item_details.itemtype;

END LOOP;

UPDATE wf_notifications
SET    status = 'CLOSED'
,      end_date = SYSDATE
,      user_comment = 'Manual transition'
WHERE  notification_id = r_current_details.nid;

WF_ITEM_ACTIVITY_STATUS.create_status ( itemtype => r_item_details.itemtype
                                       , itemkey  => p_itemkey
                                       , actid    => v_actid
                                       , status   => 'COMPLETE'
                                       , result   => 'Forced Completion'
                                       , beginning => SYSDATE );

WF_ENGINE.HandleError ( itemtype => r_item_details.itemtype
                       , itemkey  => p_itemkey
                       , activity =>
r_item_details.process_name||':'||r_item_details.instance_label
                       , command  => 'RETRY' );

EXCEPTION
WHEN e_missing_parameters THEN
    RAISE_APPLICATION_ERROR ( e_missing_parameters#
                              , 'Missing mandatory parameters in call to WF_JUMP' );

WHEN e_missing_actid THEN
    RAISE_APPLICATION_ERROR ( e_missing_actid#
                              , 'Activity ID could not be found' );

```

```
    WHEN OTHERS THEN  
        RAISE;  
END wf_jump;  
/
```

Copyright and License

This document, the code samples contained therein, and any associated code, samples or other documentation, remains copyright TS Fifteen Ltd. The moral rights of Matthew Searle to be identified as the author of the document have been asserted.

This code and document may be freely used and distributed, providing that it remains unchanged, and includes the author name and company information. This document must not be changed without approval from the copyright holder.

Disclaimer and Limitation of Liability

TS Fifteen Ltd reserves the right to make changes or updates to this software at any time, without notice.

Except where expressly provided otherwise, all content, materials, information, software, products and services provided, are provided on an "as is" and "as available" basis. TS Fifteen Ltd expressly disclaims all warranties of any kind, whether express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose and non-infringement. TS Fifteen Ltd makes no warranty that: (a) the results that may be obtained from the use of the software will be accurate or reliable; or (b) the quality of any information, or other material obtained by you will meet your expectations.

Any content, materials, information or software downloaded or otherwise obtained is done at your own discretion and risk. TS Fifteen Ltd shall have no responsibility for any damage to your computer system or loss of data that results from the download of any content, materials, information or software.

In no event shall TS Fifteen Ltd be liable for any direct, indirect, incidental, special or consequential damages, or damages for loss of profits, revenue, data or use, incurred by you or any third party, whether in an action in contract or tort, arising from your access to, or use of, this software.

Some jurisdictions do not allow the limitation or exclusion of liability. Accordingly, some of the above limitations may not apply to you.