

PL/Flow User guide

Table of contents

1	Introduction	2
1.1	The Oracle demo user SCOTT	2
1.2	Adding PL/FLOW to SCOTT's schema	2
2	Example 1: requesting a bonus	4
2.1	The process in PL/FLOW process definition tables	4
2.2	Adding SCOTT's company's members to WF_PARTICIPANTS	7
2.3	Running the example process	8
3	More examples	9
3.1	Subflows: top process 20	9
3.2	Subflows: sub process 30	10
3.3	Running the process	12
3.4	Parameter passing	12
3.4.1	Actual parameters vs formal parameters	12
3.4.2	Parameter declaration in PL/Flow.....	13
3.4.3	Parameter passing in PL/Flow	13
3.5	Synchronous vs asynchronous subprocess execution.....	14
4	PL/Flow in an application	15
4.1	Why use a workflow engine?	15
4.2	What to put in PL/FLOW and what to put in the normal tables?.....	15
5	Example program source	16
5.1	example1_run_flow.sql	16

1 Introduction

To illustrate how to enter a process definition into PL/FLOW, an example workflow process will be added to Oracle's demo user SCOTT. 2003 12 01: This example currently only runs against the PL/FLOW that is in CVS. A dump file containing PL/FLOW and this example is available for easy download.

1.1 The Oracle demo user SCOTT

The db schema SCOTT¹ contains the following tables:

BONUS

ENAME	JOB	SAL	COMM
-------	-----	-----	------

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-12-1980	800		20
7499	ALLEN	SALESMAN	7698	20-2-1981	1600	300	30
7521	WARD	SALESMAN	7698	22-2-1981	1250	500	30
7566	JONES	MANAGER	7839	2-4-1981	2975		20
7654	MARTIN	SALESMAN	7698	28-9-1981	1250	1400	30
7698	BLAKE	MANAGER	7839	1-5-1981	2850		30
7782	CLARK	MANAGER	7839	9-6-1981	2450		10
7788	SCOTT	ANALYST	7566	19-4-1987	3000		20
7839	KING	PRESIDENT		17-11-1981	5000		10
7844	TURNER	SALESMAN	7698	8-9-1981	1500	0	30
7876	ADAMS	CLERK	7788	23-5-1987	1100		20
7900	JAMES	CLERK	7698	3-12-1981	950		30
7902	FORD	ANALYST	7566	3-12-1981	3000		20
7934	MILLER	CLERK	7782	23-1-1982	1300		10
7369	SMITH	CLERK	7902	17-12-1980	800		20

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

This schema contains information about the departments and employees of a company. Each employee has a salary, a job and, if the employee is not the president, a manager.

1.2 Adding PL/FLOW to SCOTT's schema

Change to the directory where PL/FLOW was unzip/tarred:

¹ Oracle's default installation contains the user SCOTT with password TIGER.

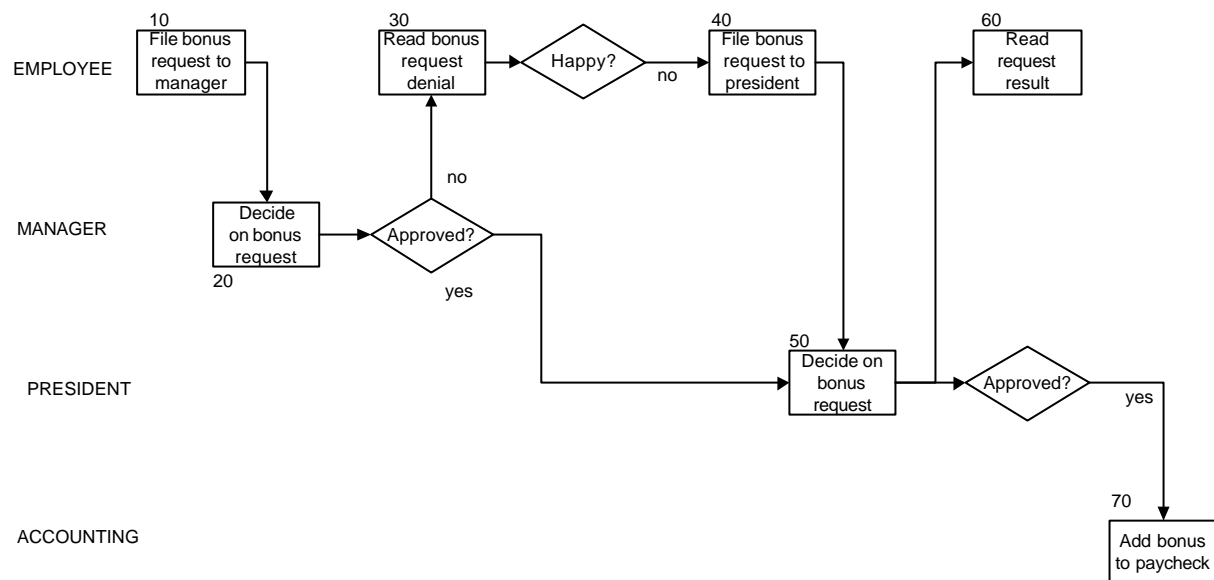
Run the following commands:

```
sqlplus scott/tiger@yourtnsname @plflowddl.sql  
sqlplus scott/tiger@yourtnsname @plflowrepstuff.sql  
sqlplus scott/tiger@yourtnsname @plflow.pks  
sqlplus scott/tiger@yourtnsname @plflow.pkb
```

You can also create a new Oracle user, grant this user select rights to SCOTT's tables, and then load PL/FLOW into this user's schema. This is what the script **install.sh** does.

2 Example 1: requesting a bonus

For the example process, let's assume that each of these employees can file a request for a bonus. The employee's manager should check this bonus request. If the manager agrees with the bonus request, the request is forwarded to the president. If the manager denies the request, the employee is notified and has the option to complain to the president, or he can leave it at that. The president's decision is final and irrevocable. If a bonus needs to be paid, the accounting department needs to change the salary check.



When does the flow start, and when does it end? The activity that has no incoming transitions is the start activity. If there is a loop to the start activity, then each transition has incoming transitions. If that is the case, the activity with the lowest ID is selected as start activity. When on completion of an activity instance (workitem) there is no transition to be done to another activity, the process instance will be completed.

2.1 The process in PL/FLOW process definition tables

This process definition can be represented by the following data in the PL/FLOW process definition tables:

WF_PARTICIPANTS

ID	NAME	DESCRIPTION	PARTICIPANT_TYPE
10	EMPLOYEE	The employee role	ROLE
20	MANAGER	The manager role	ROLE
30	PRESIDENT	The president role	ROLE
40	ACCOUNTING	The accounting role	ROLE

Because ID numbers are entered manually, I personally like to number them like old basic. In that case, when new process definition data is to be entered 'between' current data, it is possible to number the new stuff 'between' the current stuff.

PARTICIPANT_TYPE can be one of 'ROLE', 'ORGANIZATIONAL_UNIT', 'HUMAN' and 'SYSTEM'. A process definition only contains ROLES; the other participant types will be used later on.

WF_PROCESSES

ID	NAME	DESCRIPTION
10	FILE BONUS REQUEST	The process of filing a request for a bonus by an employee.

Optional information that can be specified for a process (author, version, creation date, duration, limit, valid from, valid to, working time and waiting time) is omitted in this example.

There is no entity in PL/FLOW table representing the concept of 'Package' (see [1] page 19). Grouping of related processes is done by listing them in the same Oracle user's schema. A 'package' is used to 'group' the processes, participants, applications and data fields together. In PL/FLOW these are top-level (no foreign keys to other tables) tables, so the Oracle schema PL/FLOW is loaded in acts like a 'package'.

WF_ACTIVITIES

PRCE_ID	ID	PATI_ID	NAME	DESCRIPTION	JOIN	SPLIT	CREATE_DELAY	START_MODE	FINISH_MODE	IMPLEMENTATION
10	10	10	File request to manager	File request to manager	-	-	0	MANUAL	MANUAL	NO
10	20	20	Decide on bonus request	Decide on bonus request	-	XOR	0	MANUAL	MANUAL	NO
10	30	= performer of acti 10	Read bonus request denial	Read bonus request denial	-	XOR	0	MANUAL	MANUAL	NO
10	40	= performer of acti 30	File bonus request to president	File bonus request to president	-	-	0	MANUAL	MANUAL	NO
10	50	30	Decide on bonus request	Decide on bonus request	XOR	AND	0	MANUAL	MANUAL	NO
10	60	= performer of acti 10	Read request result	Read request result	-	-	0	MANUAL	MANUAL	NO
10	70	40	Add bonus to paycheck	Add bonus to paycheck	-	-	0	MANUAL	MANUAL	NO

- In this table the column PATI_ID contains the foreign key to the PARTICIPANT record that contains the role. In the first row, PATI_ID is 10, which is the ID of the role 'EMPLOYEE'.
- Activities that have no more than one incoming (join) or outgoing (split) activities have a *don't care* transition type, when the transition is unconditional. A don't care transition type is entered into the database as NULL. This kind of transition type is not allowed when there are multiple incoming or outgoing transitions, and this is checked by PL/FLOW at runtime.

- Since there is no application yet for filing and deciding on bonus requests, all activities in this process model have a MANUAL start and finish mode, and NO implementation. An application will be added later on in this example.
- Activities 30, 40 and 60 have a value in the 'ASSIGN_TO' attribute, this is not mentioned in the table above. ASSIGN_TO is used to determine an assigned performer dynamically on runtime. The activities 30,40 and 60 should be performed by the same EMPLOYEE that originally filed the bonus request and nobody else. To tell this to workflow, an SQL query that returns the correct participant ID can be put in the 'ASSIGN_TO' attribute. The value of ASSIGN_TO for the activities 30,40 and 60 is the string:

```

select pati_id
  from wf_performers
 where state = 'CURRENT'
    and acin_id =
(select id
  from wf_activity_instances
 where acti_prce_id=10
    and acti_id=10
    and prin_id = :prin_id_in)

```

Queries put in ASSIGN_TO should always have one bind parameter :prin_id_in, which is the process instance id.

WF_TRANSITIONS

ACTI_PRCE_ID_FROM	ACTI_ID_FROM	ACTI_PRCE_ID_TO	ACTI_ID_TO	NAME	DESCRIPTION	CONDITION	CONDITION_TYPE
10	10	10	20	request to manager	request to manager		
10	20	10	30	denied by manager	denied by manager	a.name='APPROVED' AND i.value <>'Y'	CONDITION
10	20	10	50	approved by manager	approved by manager	OTHERWISE	CONDITION
10	30	10	40	not happy	employee is not happy	a.name='HAPPY' and i.value='N'	CONDITION
10	40	10	50	request to president	request to president		
10	50	10	60	result to employee	result to employee		
10	50	10	70	change paycheck	change paycheck	a.name='APPROVED' and i.value='Y'	CONDITION

- Activity 10 30 can be the end of the process instance, when the employee is happy. A process instance is marked 'COMPLETED' when on activity instance completion there are no transitions to other activities.
- Activity 10 50 has multiple outgoing (split) transitions, so the split transition type must be specified. Because the paycheck only needs to be changed when the request is approved, the split type must be a 'conditional AND'. The transition from activity 10 50 to 10 60 should always occur. By leaving the condition empty, it will default to true, which means the transition will always take place.

- An easy way to check if all transitions are listed is by counting all the arcs to an activity, excluding the arcs to a decision point. There are seven.

WF_ATTRIBUTES

ID	PRCE_ID	DATA_TYPE	NAME	LENGTH	DESCRIPTION	INITIAL_VALUE	KEEP
10	10	INTEGER	EMPNO	10	Link to EMP.EMPNO		Y
20	10	CHARACTER	APPROVED	1	Is the request approved?		N
30	10	CHARACTER	HAPPY	1	Is the employee happy with de result?		N

The attributes values with KEEP=N are deleted on process instance completion. The amount of money is not workflow relevant data, because it is not used to identify an object (like a 'the employee filing the bonus') and it is not used in a condition for a transition.

2.2 Adding SCOTT's company's members to WF_PARTICIPANTS

Now the process definition is filled, PL/FLOW needs to be told about the people that work at SCOTT's company. Instead of manually adding records, this operation can be done with a few SQL statements.

Each employee becomes a participant with PARTICIPANT_TYPE = 'HUMAN'.

```
INSERT INTO wf_participants
  ( id, name, description, participant_type )
  SELECT empno, ename, job, 'HUMAN'
  FROM scott.emp
```

Each employee is granted the role emp.

```
INSERT INTO wf_participant_relations
  ( pati_id_arg1, pati_id_arg2, relation_type )
  SELECT id, 10, 'GRANT'
  FROM wf_participants
  WHERE id IN (SELECT empno FROM scott.emp) -- only the emps
```

Grant the managers the role MANAGER

```
INSERT INTO wf_participant_relations
  ( pati_id_arg1, pati_id_arg2, relation_type )
  SELECT id, 20, 'GRANT'
  FROM wf_participants
  WHERE id IN (SELECT empno
                FROM scott.emp
                WHERE job='MANAGER') -- only the managers
```

Grant the 'PRESIDENT' role to the president

```
INSERT INTO wf_participant_relations
  ( pati_id_arg1, pati_id_arg2, relation_type )
  SELECT id, 30, 'GRANT' -- 30 is id of participant role PRESIDENT
```

```

FROM wf_participants
WHERE id IN (SELECT empno
             FROM scott.emp
             WHERE job='PRESIDENT') -- only the presidents

```

Create the department that does accounting

```

INSERT INTO wf_participants
  ( id, name, description, participant_type )
VALUES
  ( 1, 'ACCOUNTING', 'The accounting department', 'ORGANIZATIONAL_UNIT' )

```

Grant the accounting role to the accounting department

```

INSERT INTO wf_participant_relations
  ( pati_id_arg1, pati_id_arg2, relation_type )
VALUES ( 1, 40, 'GRANT' )

```

Finally, make the accounting employees 'member of' the accounting organization.

```

INSERT INTO wf_participant_relations
  ( pati_id_arg1, pati_id_arg2, relation_type )
SELECT 1, id, 'MEMBER OF'
FROM wf_participants
WHERE id IN (SELECT empno -- the id's from accounting emps
            FROM scott.emp
            WHERE deptno=
              (SELECT deptno -- get id from accounting
               FROM scott.dept
               WHERE dname='ACCOUNTING'))

```

Please note: It is absolutely coincidental that the names of the PL/FLOW roles match with some of the job names in SCOTT's EMP table. The PL/FLOW rolename isn't used anywhere, only the ID of the role. The names of the PL/FLOW roles could as well have been 'abbrqz', 'wrlvbnm' or '&*()(*&', but that would make a bad example.

The reason why the other departments besides 'ACCOUNTING' are not mentioned as organizational unit, is that the other departments are not playing a role in the process definition. It is also possible to omit the 'ACCOUNTING' organizational unit, and just grant the 'ACCOUNTING' role to each of the 3 employees working at 'ACCOUNTING'. The benefit of granting roles to an organizational unit is obvious when an organization is granted more than just one kind of role. Then, when a new employee starts working, the only thing that needs to be done is telling PL/FLOW which organizational unit the employee is member of.

2.3 Running the example process

The file 'example1_run_flow.sql' (see section 5.1 for the program source) is a SQL script containing several PL/SQL anonymous block that call the PL/SQL api. In normal use, these kind of calls to the PL/FLOW client API should be added in an application (or script), on the start and end of a the application.

Since there is no application yet, the SQL script performs the following events:

1. Scott files a bonus request
2. The manager JONES denies the request
3. Scott is not happy

- Scott files a bonus request to the PRESIDENT
4. The PRESIDENT approves the request
 5. a Scott reads the final result
 - b The accountant MILLER changes the paycheck

At the end of the example, at the last call to ChangeActivityInstanceState, PL/FLOW notices it is the last workitem of the process, it completes the process instance, and performs a number of clean-up activities:

1. All attributes with KEEP='N' are deleted.
2. All remaining instance data is moved to _ARCH tables.

The last action has as benefit that especially the attribute instance tables grow very fast in big applications. For the workflow engine, the old data is not important and only slows down all the operations. For example, take a look at the following count of workitems grouped by their state.

ACTIVITY_INSTANCES.STATE	COUNT(*)
COMPLETED	82374
NOTRUNNING	3261
RUNNING	504
TERMINATED	9411
ABORTED	0

This data was taken from a production database that has run for almost two years now. It clearly shows that the current working data, the workitems with state 'RUNNING' or 'NOTRUNNING' is about 4% of the total amount of workitems.

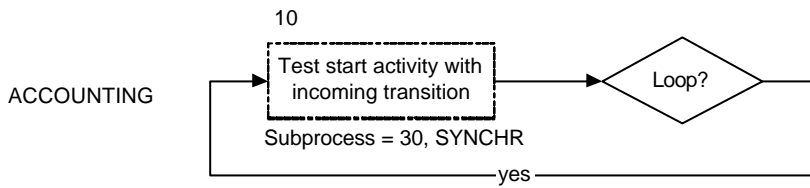
3 More examples

These are short examples for testing of subflows and deadlines

3.1 Subflows: top process 20

This process demonstrates two new things:

1. The following process has a loop to the first activity. In this case, the process definition contains no activity with no incoming transitions. To determine which activity is the start activity in this case, PL/FLOW chooses the activity with the lowest ID number.
2. The only activity in this process is further refined by a subprocess (see the next section for a description of this subprocess). The subprocess is executed *synchronous*, which means that execution of the calling process is *suspended* until the child process is completed.



WF_PROCESSES

ID	NAME	DESCRIPTION
20	Example process 20	Example process 20 for testing subflow call and start activity with incoming transition

WF_ATTRIBUTES

ID	PRCE_ID	DATA_TYPE	NAME	LENGTH	DESCRIPTION	INITIAL_VALUE	KEEP
40	20	CHARACTER	LOOP?	1	Loop?		N

WF_ACTIVITIES

PRCE_ID	ID	PRCE_ID_HAS_SUBFLOW	PATI_ID	NAME	DESCRIPTION
20	10	30	40	Test start activity with incoming transition	Test start activity with incoming transition

JOIN	SPLIT	CREATE_DELAY	START_MODE	FINISH_MODE	IMPLEMENTATION	SUBFLOW_EXECUTION
-	-	0	AUTOMATIC	AUTOMATIC	SUBFLOW	SYNCHR

WF_ACTUAL_PARAMETERS

ID	ACTI_PRCE_ID	ACTI_ID	ATRI_ID	NAME
10	20	10	40	RESULTCODE

This record says that on start of activity 20,10 (which is a subprocess), parameter RESULTCODE (see the 'formal_parameters' of process 30) is linked to process attribute 40: LOOP?. Since the formal parameter RESULTCODE is an OUT parameter, the effect is that on completion of the subprocess, attribute 40 (LOOP?) is filled with the value of the RESULTCODE parameter of the subprocess.

WF_TRANSITIONS

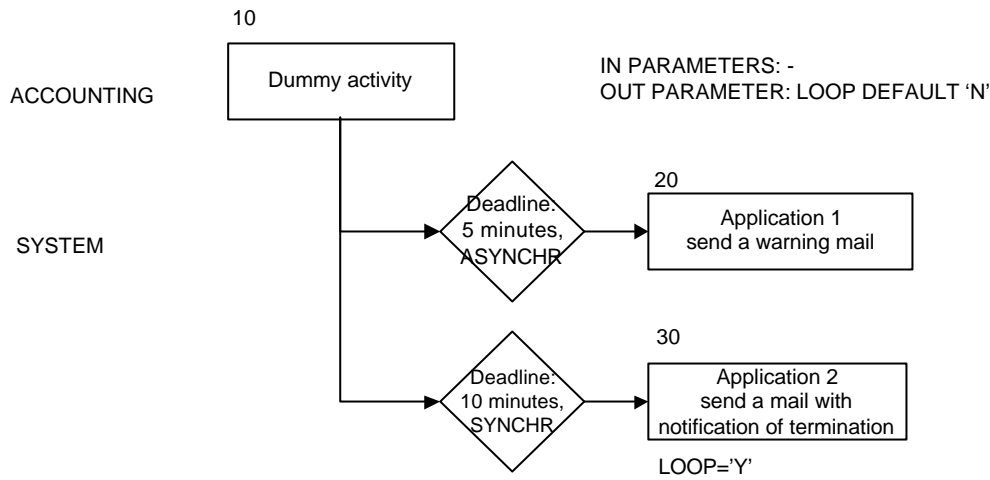
ACTI_PRCE_ID_FROM	ACTI_ID_FROM	ACTI_PRCE_ID_TO	ACTI_ID_TO	NAME	DESCRIPTION	CONDITION	CONDITION_TYPE
20	10	20	10	loop to start activity	loop to start activity	a.name='LOOP?' AND i.value='Y'	CONDITION

3.2 Subflows: sub process 30

This process is subflow of process 20 activity 10. This process also illustrates some new features:

1. parameter passing between a parent and a child process.
2. asynchronous and synchronous deadlines
3. multiple deadlines on one activity
4. activity implementation 'TOOL' (calling an application)

5. using the pl_flow.sendmail function



WF_PROCESSES

ID	NAME	DESCRIPTION
30	Example process 30	Example process 30 is to be called as subflow by process 20

WF_ATTRIBUTES

ID	PRCE_ID	DATA_TYPE	NAME	LENGTH	DESCRIPTION	INITIAL_VALUE	KEEP
50	30	CHARACTER	RESULTCODE	1	Resultcode. Y means loop the parent process	Y	N

WF_FORMAL_PARAMETERS

ID	ACTI_PRCE_ID	ATRI_ID	DATA_TYPE	FOPA_MODE	DESCRIPTION
10	30	50	CHARACTER	OUT	This process attribute is OUT parameter

WF_APPLICATIONS

ID	NAME	DESCRIPTION	PLSQL_PROC_NAME
10	Send a warning mail	Send a warning mail	SEND_WARNING_MAIL
20	Send notification of termination	Send notification of termination	SEND_TERMINATION_NOTIFICATION

WF_ACTIVITIES

PRCE_ID	ID	APLI_ID	PATI_ID	NAME	DESCRIPTION
30	10		40	Dummy activity	The first dummy activity. With a deadline
30	20	10	1	Application 10: send a warning mail	Application 10: send a warning mail
30	30	20	1	Application 20: send termination notification	Application 20: send termination notification

JOIN	SPLIT	CREATE_DELAY	START_MODE	FINISH_MODE	IMPLEMENTATION	SUBFLOW_EXECUTION
-	-	0	MANUAL	MANUAL		
-	-	0	AUTOMATIC	AUTOMATIC	TOOL	
-	-	0	AUTOMATIC	AUTOMATIC	TOOL	

WF_DEADLINES

ID	ACTI_ID	ACTI_PRCE_ID	EXECUTION	CONDITION	EXCEPTION_NAME
10	10	30	ASYNCHR	1/24/60	dummy_activity_time_warning
10	10	30	SYNCHR	1/24/6	dummy_activity_timed_out

WF_TRANSITIONS

ACTI_PRCE_ID_FROM	ACTI_ID_FROM	ACTI_PRCE_ID_TO	ACTI_ID_TO	NAME	DESCRIPTION	CONDITION	CONDITION_TYPE
30	10	30	20	deadline warning transition	deadline timeout transition	dummy_activity_time_warning	EXCEPTION
30	10	30	30	deadline timeout transition	deadline timeout transition	dummy_activity_timed_out	EXCEPTION

3.3 Running the process

When a process instance for the top process is created and then started, the first activity will be created. Since this activity has start_mode 'AUTOMATIC', which is kind of the only correct start_mode for an activity with implementation 'SUBFLOW', it will be started automatically. This results in the creation of another process instance. This process instance is also started, creating the first activity instance 'dummy activity'. The 'calling' activity is set to state 'SUSPENDED' by the participant PL/FLOW, because the subflow execution type is 'SYNCHR'.

ACTI_PRCE_ID	ACTI_ID	PRIN_ID	ID	STATE	DATE_CREA	DATE_STAR	DATE_ENDE
20	10	2	17	SUSPENDED	09-DEC-03	09-DEC-03	
30	10	7	19	NOTRUNNING	09-DEC-03		

... to be finished

3.4 Parameter passing

3.4.1 Actual parameters vs formal parameters

Quoted from Oracle's PL/SQL Users guide:

“Subprograms pass information using parameters. The variables or expressions referenced in the parameter list of a subprogram call are actual parameters. For example, the following procedure-call lists two actual parameters named emp_num and amount:

```
raise_salary(emp_num, amount);
```

The next procedure call shows that expressions can be used as actual parameters:

```
raise_salary(emp_num, merit + cola);
```

The variables declared in a subprogram spec and referenced in the subprogram body are formal parameters. For example, the following procedure declares two formal parameters named `emp_id` and `amount`:

```
PROCEDURE raise_salary (emp_id INTEGER, amount REAL) IS
    current_salary REAL;
    ...
BEGIN
    SELECT sal INTO current_salary FROM emp WHERE empno = emp_id;
    ...
    UPDATE emp SET sal = sal + amount WHERE empno = emp_id;
END raise_salary;
```

A good programming practice is to use different names for actual and formal parameters.

When you call procedure `raise_salary`, the actual parameters are evaluated and the result values are assigned to the corresponding formal parameters. If necessary, before assigning the value of an actual parameter to a formal parameter, PL/SQL converts the datatype of the value. For example, the following call to `raise_salary` is valid:

```
raise_salary(emp_num, '2500');
```

The actual parameter and its corresponding formal parameter must have compatible datatypes. For instance, PL/SQL cannot convert between the `DATE` and `REAL` datatypes. Also, the result value must be convertible to the new datatype. The following procedure call raises the predefined exception `VALUE_ERROR` because PL/SQL cannot convert the second actual parameter to a number:

```
raise_salary(emp_num, '$2500'); -- note the dollar sign
```

3.4.2 Parameter declaration in PL/Flow

The concepts in PL/Flow that can be called with parameters are *tools* (like stored procedures, or external calls to e.g. a webservice) and *processes*. The declaration of the parameters is done by listing them in the table `WF_FORMAL_PARAMETERS`.

The WfMC standard doesn't incorporate the concept of named parameter passing. The only way to specify a parameter is by its position in the parameter list: the index. In my opinion it was very strange that even in the declaration of variables of a tool or process no names were used, so I extended the `formal_parameters` table with the attribute 'name'.

NB. Note that both 'index' and 'mode' are reserved words in Oracle. That's why Oracle designer transforms the names of these attributes into `FOPA_MODE` and `FOPA_INDEX`.

3.4.3 Parameter passing in PL/Flow

When a Tool or Subprocess is called, PL/Flow determines values to be passed as parameters to the called tool by reading the table `WF_ACTUAL_PARAMETERS`. An actual parameter can be specified as an expression, or as reference to a process attribute. If an actual parameter refers to an attribute, the 'expression' attribute is ignored. In other words, specify either an attribute or an expression in `formap` parameters. The expression is evaluated at runtime by doing a

```
SELECT <expression>
FROM WF_ACTIVITY_INSTANCES acin
WHERE acin.id=:acin_id_in
USING <value of the activity instance id>
```

This allow you to use expressions like

- 1+1
- SYSDATE
- myseq.NEXTVAL

And you can also get some value from the current calling activity like this:

- date_due
- ```
(select name
 from wf_activities
 where id=acin.acti_id
 and prce_id=acin.acti_prce_id)
```

Since date\_due is an attribute of wf\_activity\_instances, it can be used in as expression in the SQL query mentioned above.

Note that if the only thing you do is get the value of an attribute instance, it's easier to just list atri\_id=2000 in the actual parameter table.

### **3.5 Synchronous vs asynchronous subprocess execution**

Fancy names for an easy concept. Synchronous execution means 'call' and asynchronous means 'fork' / 'spawn'. Synchronous waits for the subprocess to be completed before the parent process continues. Asynchronous doesn't wait. Asynchronous subprocesses can't have OUT parameters.

## 4 PL/Flow in an application

### 4.1 Why use a workflow engine?

At this point you might wonder 'why use a workflow engine? why not just create a status attribute for the 'bonus requests' entity?'. This is a very important question, and it should be asked in every application where using a workflow engine is considered. For very simple and small applications, adding a workflow engine might be just too much. I am not going to make a plea for using a workflow engine, I'm just going to list a few things you get 'for free' and are there to stay once you've implemented a workflow:

If the application you are designing is accessible to multiple people, it's most likely you can draw a process diagram like the 'file bonus request' diagram. It's also likely that the different people act under different roles. If you can then identify clear and separated activities, along with the applications people use when doing those activities, the most difficult part of using a workflow engine is already finished. What do you get when you decide to list the roles and activities in PL/FLOW:

- an authorisation switch: who may do what?
- resource assignment software
- management information: how long does which activity take, what are bottlenecks?
- a way or method to divide the whole application in a uniform way using industry standard vocabulary

### 4.2 What to put in PL/FLOW and what to put in the normal tables?

This is a difficult question. I like to use the following guidelines:

- PL/FLOW and the normal tables should be completely separated. When a kind of information is essential to an application, it should be stored in the 'normal' tables. Even though PL/FLOW stores a list of participants, if information about persons is essential to your application, do not get tempted to only use the PL/FLOW participants table to list these persons. The workflow 'view' of the world is a different 'view' than your applications view, and therefore a participant and your 'person' are different concepts. They should be modelled separated. To link a workflow participant to the 'normal' data, the EXTERNAL\_REFERENCES table should be used.

- PL/FLOW contains information to decide what should be done, when, and by whom. If an application needs to display historic info (like 'patient record last changed on <date>'), this is could be called an essential characteristic of the modeled domain. In a medical application, records of who did what when should be kept. It is an idea to just do a select in the activity instances table, but this is, IMHO, also wrong. The application should not lose data when PL/FLOW tables are removed. Therefore, if information about acts and logging is essential to your application, do not rely on the PL/FLOW data structure for primary information about it. Implement it as if PL/FLOW was not there. PL/FLOW is for 'steering' the applications activities. The 'normal' tables are for 'storing' data about the world, needed for the specific application.

- For management information the PL/FLOW instance tables will be very useful. Wait, you say, if showing management information like how many this or how fast this stuff is done is

part of the initial requirements of my application, shouldn't the necessary information be modeled along with all the rest of the application, and not use PL/FLOW for that? The answer is: not really. Just like PL/FLOW is something outside your primary application data, even more so is management information. Information about how fast you do something, or how often, says something *about* the something, it is like meta information. Therefore you could create a nice application, forget the management information for a while, design the applications database structure and design the process, build the application, and then, last, build some views on top of the PL/FLOW activity instance tables. Then end result is that different concerns have become different parts of your application.

## 5 Example program source

### 5.1 example1\_run\_flow.sql

The following pieces of code are part of the script  
 <plflowdir> /examples/example1\_run\_flow.sql

```

-- Step 1:
-- Scott creates a process instance and files a bonus request.

DECLARE
 l_prin_id WF_PROCESS_INSTANCES.ID%TYPE; -- ID of new created process instance
 l_pati_id_manager WF_PARTICIPANTS.ID%TYPE; -- ID of participant 'SCOTT'
 l_acin_id WF_ACTIVITY_INSTANCES.ID%TYPE; -- ID of new created activity instance
BEGIN
 -- create the process.
 -- I get a new sequence number
 SELECT prin_seq.NEXTVAL
 INTO l_prin_id
 FROM DUAL;
 -- 2 create the process
 PL_FLOW.CreateProcessInstance(
 prce_id_in=>10, -- 10 is process id of file bonus request process
 prin_id_in=>l_prin_id
);

 -- Scott is the emp wanting a bonus
 SELECT id
 INTO l_pati_id_manager
 FROM wf_participants
 WHERE name='SCOTT';

 -- The attribute is named 'EMPNO'
 -- scotts EMPNO is 7788
 -- note that because the attribute instances values are not constrained
 -- in anyway to actually contain an existing EMPNO in the SCOTT.EMP table.
 PL_FLOW.AssignProcessInstanceAttribute(
 prin_id_in =>l_prin_id,
 name_in =>'EMPNO',
 value_in =>l_pati_id_manager
);

 -- Start the process instance
 -- The process is started by the participant 'SCOTT'

 PL_FLOW.StartProcess(
 prin_id_in => l_prin_id, -- the previously created process instance
 pati_id_in => l_pati_id_manager
);

 -- At this point, there is a workitem in this process
 -- that represents the 'File request to manager' workitem
 -- Every EMP may process it, but SCOTT is the one to start it.
 -- First, find the ID of this newly created activity instance

```

```

SELECT id
 INTO l_acin_id
 FROM wf_activity_instances
 WHERE prin_id = l_prin_id -- only from this process instance
 AND acti_id = 10; -- and only activity 10. (file bonus request)

-- Now, tell PL/FLOW that SCOTT starts with this workitem
PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_acin_id,
 state_in => 'RUNNING',
 pati_id_in => l_pati_id_manager -- pati id of SCOTT
);

-- Now, complete the same workitem
PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_acin_id,
 state_in => 'COMPLETED',
 pati_id_in => l_pati_id_manager -- pati id of SCOTT
);
END;
/
Prompt Activity instances after SCOTT completed 'file bonus request'
SELECT acti_prce_id, acti_id, prin_id, id, state, date_created, date_started, date_ended FROM
WF_ACTIVITY_INSTANCES
/

-- Step 2:
-- The manager 'JONES'
-- Now fetch the worklist of the manager jones.

DECLARE
 l_prin_id WF_PROCESS_INSTANCES.ID%TYPE; -- ID of new created process instance
 l_pati_id_manager WF_PARTICIPANTS.ID%TYPE; -- ID of participant 'JONES'
 l_acin_id WF_ACTIVITY_INSTANCES.ID%TYPE; -- ID of new created process instance

 l_worklist_cursor PL_FLOW.generic_curtype; -- a cursor variable

 -- worklist_rowtype is a view which only purpose is to be used for %ROWTYPE.
 l_worklist_record worklist_rowtype%ROWTYPE;

 l_dummy_int PLS_INTEGER;

BEGIN
 -- At this point, there is a workitem on the worklist of the managers
 -- The manager is JONES, with pati_id 7566
 SELECT id
 INTO l_pati_id_manager
 FROM wf_participants
 WHERE name='JONES';

 PL_FLOW.OpenWorkList(
 pworklist_filter => 'STATE='NOTRUNNING'',
 pati_id_in => l_pati_id_manager, -- this is the ID of jones
 count_flag => 0, -- should rowcount be returned?
 pquery_handle => l_worklist_cursor,
 pcount => l_dummy_int
);
 -- get the workitem for this manager
 -- please note that in this example, assumptions are being made about what this worklist
 -- query will return. In real life, the worklist will be displayed on screen
 FETCH l_worklist_cursor INTO l_worklist_record;
 CLOSE l_worklist_cursor;

 -- Now, tell PL/FLOW that JONES starts with this workitem
 PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'RUNNING',
 pati_id_in => l_pati_id_manager -- pati id of JONES
);

 -- Jones decides to deny the request.
 PL_FLOW.AssignProcessInstanceAttribute(
 prin_id_in => l_worklist_record.prin_id,
 name_in => 'APPROVED',
 value_in => 'N'
);

```

```

);

-- Now, complete the same workitem
PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'COMPLETED',
 pati_id_in => l_pati_id_manager -- pati id of JONES
);
END;
/

Prompt Activity instances after JONES completed 'decide on bonus request'
SELECT acti_prce_id, acti_id, prin_id, id, state, date_created, date_started, date_ended FROM
WF_ACTIVITY_INSTANCES
/

Prompt Performers after JONES completed 'file bonus request': 7788 is SCOTT.
SELECT id, pati_id, acin_id, date_created, state, accepted FROM WF_PERFORMERS
/

-- Step 3:
-- The bonus was disapproved, so SCOTT will have a request denial on HIS worklist.
-- Because SCOTT is assigned the last workitem, he is the only one that may start it.

DECLARE
 l_prin_id WF_PROCESS_INSTANCES.ID%TYPE;
 l_pati_id_scott WF_PARTICIPANTS.ID%TYPE;
 l_acin_id WF_ACTIVITY_INSTANCES.ID%TYPE;

 l_worklist_cursor PL_FLOW.generic_curtype; -- a cursor variable

 -- worklist_rowtype is a view which only purpose is to be used for %ROWTYPE.
 l_worklist_record worklist_rowtype%ROWTYPE;

 l_dummy_int PLS_INTEGER;

BEGIN
 SELECT id
 INTO l_pati_id_scott
 FROM wf_participants
 WHERE name='SCOTT';

 PL_FLOW.OpenWorkList(
 pworklist_filter => 'STATE='NOTRUNNING'',
 pati_id_in => l_pati_id_scott, -- this is the ID of jones
 count_flag => 0, -- should rowcount be returned?
 pquery_handle => l_worklist_cursor,
 pcount => l_dummy_int
);

 -- get the workitem
 -- please note that in this example, assumptions are being made about what this worklist
 -- query will return. In real life, the worklist will be displayed on screen
 FETCH l_worklist_cursor INTO l_worklist_record;
 CLOSE l_worklist_cursor;

 -- Now, tell PL/FLOW that SCOTT starts with this workitem
 PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'RUNNING',
 pati_id_in => l_pati_id_scott
);

 -- Scott is not happy.
 PL_FLOW.AssignProcessInstanceAttribute(
 prin_id_in => l_worklist_record.prin_id,
 name_in => 'HAPPY',
 value_in => 'N'
);

 -- Now, complete the same workitem
 PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'COMPLETED',
 pati_id_in => l_pati_id_scott
);

```

```

-- Now there will be a new workitem for scott: file bonus request to president
PL_FLOW.OpenWorkList(
 pworklist_filter => 'STATE='NOTRUNNING'',
 pati_id_in => l_pati_id_scott, -- this is the ID of jones
 count_flag => 0, -- should rowcount be returned?
 pquery_handle => l_worklist_cursor,
 pcount => l_dummy_int
);
FETCH l_worklist_cursor INTO l_worklist_record;
CLOSE l_worklist_cursor;

-- Now, tell PL/FLOW that SCOTT starts with this workitem
PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'RUNNING',
 pati_id_in => l_pati_id_scott
);

-- Here would be code for the screen SCOTT can enter the bonus
-- request to the manager

-- Now, complete the same workitem
PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'COMPLETED',
 pati_id_in => l_pati_id_scott
);

END;
/

Prompt Activity instances after SCOTT completed 'read bonus request denial AND file bonus
request to president'
SELECT acti_prce_id, acti_id, prin_id, id, state, date_created, date_started, date_ended FROM
WF_ACTIVITY_INSTANCES
/

-- Step 4:
-- The president makes the final decision.

DECLARE
 l_prin_id WF_PROCESS_INSTANCES.ID%TYPE;
 l_pati_id_president WF_PARTICIPANTS.ID%TYPE;
 l_acin_id WF_ACTIVITY_INSTANCES.ID%TYPE;

 l_worklist_cursor PL_FLOW.generic_curtype; -- a cursor variable

 -- worklist_rowtype is a view which only purpose is to be used for %ROWTYPE.
 l_worklist_record worklist_rowtype%ROWTYPE;

 l_dummy_int PLS_INTEGER;

BEGIN
 -- At this point, there is a workitem on the worklist of the managers
 -- The manager is JONES, with pati_id 7566
 SELECT id
 INTO l_pati_id_president
 FROM wf_participants
 WHERE description='PRESIDENT';

 PL_FLOW.OpenWorkList(
 pworklist_filter => 'STATE='NOTRUNNING'',
 pati_id_in => l_pati_id_president,
 count_flag => 0, -- should rowcount be returned?
 pquery_handle => l_worklist_cursor,
 pcount => l_dummy_int
);
 -- get the first workitem from this list
 -- please note that in this example, assumptions are being made about what this worklist
 -- query will return. In real life, the worklist will be displayed on screen
 FETCH l_worklist_cursor INTO l_worklist_record;
 CLOSE l_worklist_cursor;

 -- Now, tell PL/FLOW that KING starts with this workitem

```

```

PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'RUNNING',
 pati_id_in => l_pati_id_president
);

-- King decides to approve the request.
PL_FLOW.AssignProcessInstanceAttribute(
 prin_id_in => l_worklist_record.prin_id,
 name_in => 'APPROVED',
 value_in => 'Y'
);

-- Now, complete the same workitem
PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'COMPLETED',
 pati_id_in => l_pati_id_president
);
END;
/

Prompt Activity instances after KING completed 'decide on bonus request'
SELECT acti_prce_id, acti_id, prin_id, id, state, date_created, date_started, date_ended FROM
WF_ACTIVITY_INSTANCES
/

Prompt Performers after KING completed 'decide on bonus request'
SELECT id, pati_id, acin_id, date_created, state, accepted FROM WF_PERFORMERS
/

-- Now there are two workitems, one for SCOTT, and one for the ACCOUNTING department.

-- Step 5a:
-- SCOTT reads the result.

DECLARE
 l_prin_id WF_PROCESS_INSTANCES.ID%TYPE;
 l_pati_id_scott WF_PARTICIPANTS.ID%TYPE;
 l_acin_id WF_ACTIVITY_INSTANCES.ID%TYPE;

 l_worklist_cursor PL_FLOW.generic_curtype; -- a cursor variable

 -- worklist_rowtype is a view which only purpose is to be used for %ROWTYPE.
 l_worklist_record worklist_rowtype%ROWTYPE;

 l_dummy_int PLS_INTEGER;

BEGIN
 SELECT id
 INTO l_pati_id_scott
 FROM wf_participants
 WHERE name='SCOTT';

 PL_FLOW.OpenWorkList(
acti_id=60',
 pworklist_filter => 'STATE='NOTRUNNING' '||CHR(38)||' ACTIVITIES prce_id=10 and
 pati_id_in => l_pati_id_scott, -- this is the ID of scott
 count_flag => 0, -- should rowcount be returned?
 pquery_handle => l_worklist_cursor,
 pcount => l_dummy_int
);

 -- get the workitem
 FETCH l_worklist_cursor INTO l_worklist_record;
 CLOSE l_worklist_cursor;

 -- Now, tell PL/FLOW that SCOTT starts with this workitem
 PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'RUNNING',
 pati_id_in => l_pati_id_scott
);

 -- Here would be code that displays the result on SCOTT's screen.

```

```

-- Now, complete the same workitem
PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'COMPLETED',
 pati_id_in => l_pati_id_scott
);
END;
/

Prompt Activity instances after SCOTT completed 'read final result'
SELECT acti_prce_id, acti_id, prin_id, id, state, date_created, date_started, date_ended FROM
WF_ACTIVITY_INSTANCES
/

-- Step 5b:
-- The accountant MILLER (id 7934) handles the paycheck

DECLARE
 l_prin_id WF_PROCESS_INSTANCES.ID%TYPE;
 l_pati_id_accountant WF_PARTICIPANTS.ID%TYPE;
 l_acin_id WF_ACTIVITY_INSTANCES.ID%TYPE;

 l_worklist_cursor PL_FLOW.generic_curtype; -- a cursor variable

 -- worklist_rowtype is a view which only purpose is to be used for %ROWTYPE.
 l_worklist_record worklist_rowtype%ROWTYPE;

 l_dummy_int PLS_INTEGER;

BEGIN
 SELECT id
 INTO l_pati_id_accountant
 FROM wf_participants
 WHERE name='MILLER';

 PL_FLOW.OpenWorkList(
 pworklist_filter => 'STATE='NOTRUNNING' ||CHR(38)|| ' ACTIVITIES prce_id=10 and
acti_id=70',
 pati_id_in => l_pati_id_accountant,
 count_flag => 0, -- should rowcount be returned?
 pquery_handle => l_worklist_cursor,
 pcount => l_dummy_int
);
 -- get the workitem
 FETCH l_worklist_cursor INTO l_worklist_record;
 CLOSE l_worklist_cursor;

 -- Now, tell PL/FLOW that MILLER starts with this workitem
 PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'RUNNING',
 pati_id_in => l_pati_id_accountant
);

 -- Here would be code like changing amount on next paycheck.

 -- Now, complete the same workitem
 PL_FLOW.ChangeActivityInstanceState(
 acin_id_in => l_worklist_record.acin_id,
 state_in => 'COMPLETED',
 pati_id_in => l_pati_id_accountant
);
END;
/
Prompt The process is now completed.

COMMIT
/
SPOOL OFF

--ROLLBACK
--/
QUIT
/

```

Next example:

+ deadline

+ automated task (decision by manager automated depending on salary)

+ subflow

About ROUTE and BLOCK activities

[1] Workflow Management Coalition Workflow Standard Workflow Process Definition Interface. Document Number WFMC-TC-1025 Document Status –1.0 Final Draft  
[http://www.wfmc.org/standards/docs/TC-1025\\_10\\_xpdl\\_102502.pdf](http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf)