Business Process Training

Training:Architecture:Business Process Training Worksheets

Business Process Training Worksheets

Introduction

1. Overview
2. Setting up
3. Useful Queries

Workflow Diagrams

1. Workflow Diagrams
2. Planning Application Workflow Diagram

General Concepts

1. BPM Package
2. Workbaskets
3. Business Context
4. Delegations and Process Assignments
5. Business Stages
6. Entry Stages
7. Exit Stages
8. Standard Stages
9. Moving Between Stages
10. Workbasket Actions
11. Action Data
12. Panel Actions
13. Transition Assignments
14. Transition APIs
15. LHS Workbasket Actions
16. Starting a Business Process Routine
17. External Events
18. Putting Into Practice
19. Conditional Stages
20. Sync Stages
21. Process XML
22. Signals
23. Emails

Advanced Concepts

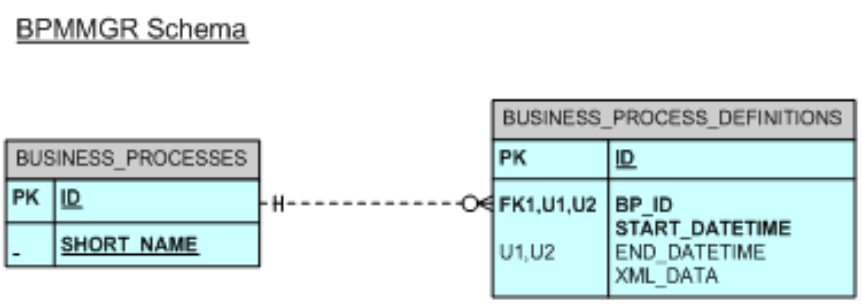
1. Subroutines
2. Context Switches
3. Contextual Assignments
4. Delay Stages
5. Parallel Processing Pattern
6. Abort Pattern
7. Orphan Stages
8. Conditional Action Sets
9. Action Overlay
10. Refreshing Stage Actions
11. On Stage Tasks
12. Using Process XML Packages

Training:Architecture:Business Process Training Worksheets:Overview

Overview

The FOX framework includes a Business Process component which can be used to create business processes in XML without the developer having to worry about how the low-level details of common tasks such as moving between stages or providing suitable actions to the correct user are carried out.

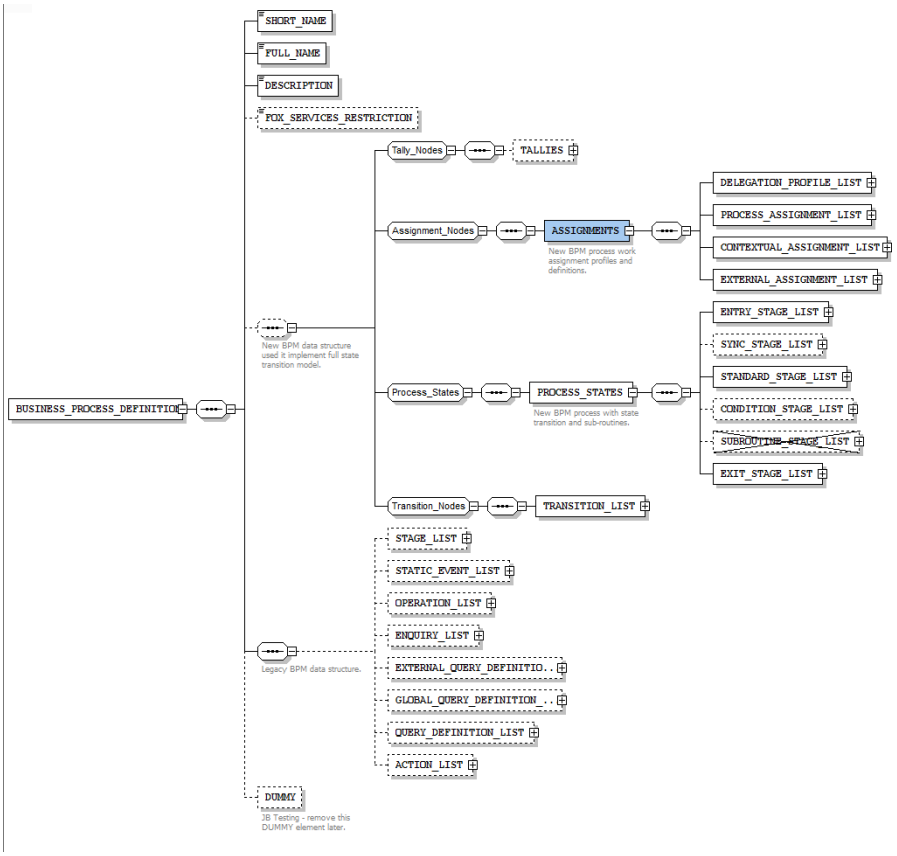
The Business Process Manager (BPM) has been implemented as a PL/SQL stored package in the database. This package BPMMGR.BPM_UPDATE, is the engine that interprets the Business Rules and provides the required services.



The Business Rules for a given application system are grouped together and defined in a Business Process Definition (BPD) XML file and stored in the BUSINESS_PROCESS_DEFINITIONS table on the database. The BPM_UPDATE PL/SQL remains static. It is the Business Process Definitions that change for each new application system.

BPD Core Elements

The Business Process Definition (BPD) is an XML file stored in the BPMMGR.BUSINESS_PROCESS_DEFINITIONS table. It must conform to the BUSINESS_PROCESS_DEFINITION.xsd XMLSchema, ask a training supervisor for the location of this document. The BPD is divided up into several sections which will be covered in this training.



The training will go into each of the core elements of the business process shown in the above diagram. These include:

- The BPD identifiers and description are covered by the elements SHORT_NAME, FULL_NAME, and DESCRIPTION.
- ASSIGNMENTS: These specify groups of people who interact with the business process.
- PROCESS_STATES: Workflow stages are implemented here. These may be used to control the flow of the business process or to provide actions to assignments allowing them to interact with the business process.
- TRANSITION_LIST: Transitions define how a business process moves between its stages.
- OPERATION_LIST: This contains an operation to start the business process.
- QUERY_DEFINITION_LIST: SQL queries and PL/SQL anonymous blocks can be defined here to be used throughout your business process.

Training:Architecture:Business Process Training Worksheets:Setting Up

Setting Up

The business process makes use of UREFs and teams, so before getting started we need to ensure that these are setup. Also during the training we will call out to some methods to perform business logic when moving between stages. These methods have been provided for you so that you do not lose focus on the the business process concepts being introduced. They will be briefly explained when they are used to ensure that you understand what is taking place.

Note: where you are required to replace *XX* with your initials please use capital letters for your initials. In most cases this does not matter as it is case in-sensitive, but where it does matter capitals are required.

If You Haven't Completed The Fox Training

If you have not completed the FOX training you will need carry out the following steps to setup the data model and module used in the Planning Application Project.

1. Data model setup
    1. Get *XX_PLAN_APP_DATA_MODEL.sql* from your BPD training directory.
    2. Replace all instances of 'xx' with your initials and run the script.
2. Package setup
    1. Get 'XX_PLAN_APP.pks' and 'XX_PLAN_APP.pkb'from your BPD training directory.
    2. Replace all instances of 'xx' with your initials and compile them on the database server.
3. Grants
    1. Get *XX_PLAN_APP_POST_PATCH_DATA_MODEL.sql* from your BPD training directory.
    2. Replace all instances of 'xx' with your initials and run the script.
4. Fox module
    1. Get *TRAINING_PLAN_APP_EDIT_V5.xml* FOX module from your FOX training directory.
    2. Replace all instances of 'xx' in the module with your initials.
    3. Rename the file to *XX_TRAINING_PLAN_APP_EDIT.xml* (where *XX* is your initials) and add the file to the envmgr.fox_components_training table.
    4. N.B. You will also need to make sure OVERLAY_POPUP_LIBRARY and PLANNING_DEC046X_WRAP exist in fox_components

UREF and Team Creation

The business process makes use of teams and UREFs. It is therefore a prerequisite that you have read and understood the following training sheets.

- UREFs
- Teams

Once you have read the training material in the above links get the *XX_TRAINING_PLAN_APP_TEAM.xml* file from your BPD training directory. Change the *XX* in the res_type and res_type_title elements in the team XML to your initials. Add a record to decmgr.resource_types with the res_type set to *XX_TRAINING_PLAN_APP_TEAM* where *XX* is your initials and set the xml_data column to your team XML.

Next run the *TEAM_SETUP_SCRIPT.sql* file located in your BPD training directory. Run the script providing the initials you used when creating your tables for the Planning Application Training system. Note you will need to run this script as XVIEWMGR.

This script will carry out the following actions:

1. Create a UREF on your plan_apps table.
2. Create a team for you to use when working with your business process consisting of applicants, viewers, case managers, documents managers, reviewers, and resource co-ordinators.
3. For each role two resource people will be created with associated web user accounts.

At the bottom of the script there is a query you can run which will show you the login id for each user created and what role they are in.

Planning Application Package

If you have completed the first section on 'haven't done fox training' then skip this section.

1. Get the *XX_PLAN_APP.pks* and *XX_PLAN_APP.pkb* files from your BPD training directory.
2. Replace all instances of *XX_* in both the spec and body with the initials you used on your planning application.
3. Compile the package spec followed by the package body.

Login Links

A business process will usually require interaction by more than one user. To simplify moving between user accounts get the *Training Login Homepage.zip* file from your BPD training directory and extract the files. Open the login.xml file for editing and change all instances of *xx* to your initials. Save your changes. Open the login.xml file in your preferred browser and add the page as a favourite. This page will allow you to quickly switch between the user accounts we created as part of your team.

Business Process Definition

1. Get *XX_TRAINING_PLAN_APP_BPD_V1.xml* from your BPD training directory.
2. Get the business_process_definition schema from BPD training directory and set XMLSPY to validate your local CodeSource\BusinessProcessDefinitions folder (or whichever folder you will be storing your business process definition in) against this schema.
3. Rename the *XX_TRAINING_PLAN_APP_BPD_V1.xml* file to use your initials and change the SHORT_NAME element within the Business Process Definition also.
4. Insert into bpmmgr.business_processes the short name and an ID (any unique ID will do).
5. Insert into bpmmgr.business_process_definitions the same ID as above in both the id and bp_id columns, sysdate in start_datetime and leave end_datetime and xml_data blank.
6. Set up clobber to clob your BPD to bpmmgr.business_process_definitions.

Training:Architecture:Business Process Training Worksheets:Useful Queries

Business Process Queries

The following queries will help when debugging business process runs. Save these files as .sql files and run the queries and APIs from TOAD. Don't worry if you do not understand them or know how to use them right away. It will become apparent what they do as the business process concepts are explained.

Current Stage and Assignments

The query contained in this file is useful for ascertaining what stages a business process is in and for each stage who it is assigned to.

Transition, Event and Stage History

This file contains three separate queries.

1. A history of stages the business process has gone through (includes ended stages)
2. A history of events raised by the business process including detailed log information of what occurred
3. A history of transitions a business process has gone through

Call Event and Rest Actions

This file has two anonymous blocks.

1. A call to new_event function. This function allows you to transition a business process from TOAD using an event label or transition alias.
2. A call to reset_process_actions. This function takes in 1 or more parameters consisting of the stage id, stage label, or business process definition name. It will recreate the actions for the given stage/business process.

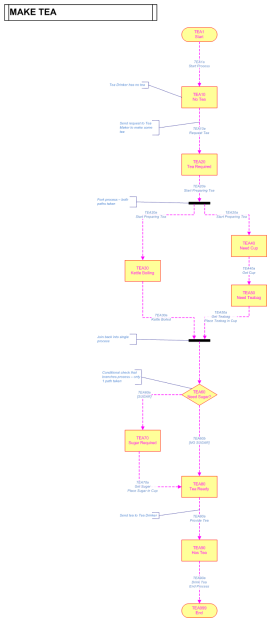
Training:Architecture:Business Process Training Worksheets:Workflow Diagrams

Workflow Diagrams

Before starting on building our business process we first need to have a clear idea of what that process is. We use Workflow Diagrams to model the business process. These diagrams are created as a result of requirements analysis and are used by the developer during implementation to create the business process.

Sample Workflow Diagram

The image below is of a Workflow Diagram showing the process that takes place when you order a cup of tea.

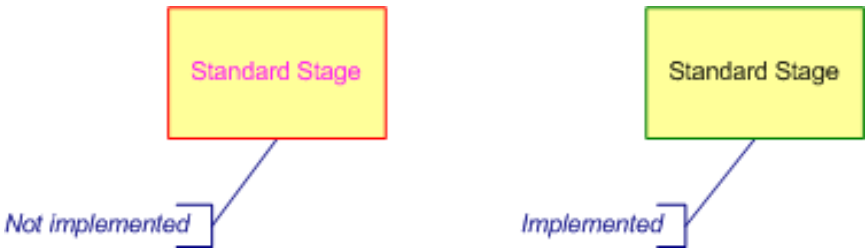


We can see from the diagram that the process of ordering a cup of tea goes through a number of stages. At each stage it waits for an action to be taken before it moves on to the next stage. Depending on what stage it is in the action may need to be taken by either the Tea Drinker or the Tea Maker.

Standards

When creating a workflow diagram the following standards must be adhered to:

1. Use standard shapes so that the diagram can be understood by all developers. These shapes can be found in your BPD training directory BPD_SHAPES.vss.
2. Stages are named with a business process mnemonic followed by a number. We generally use a non-consecutive numbering sequence for the stages with each stage number incrementing by 10. This helps for maintainability as if we need to add additional stages into the business process after it has been implemented we can keep them in sequence without having to re-number them all.
3. Transitions are marked up with the stage name followed by a letter in sequence a..z.
4. Only set the style of a shape to 'Marked Up' once it has been implemented in the business process XML. The following are examples of marked up and not marked up standard stages.



Training:Architecture:Business Process Training Worksheets:Plan App Workflow Diagram

Planning Application Business Process Diagram

Before we start get the *TRAINING_PLAN_APP_BUSINESS_PROCESS.vsd* file from BPD training directory. There are two separate business processes defined here.

The business process on the first page *Planning Application* shows the process from creating a planning application through to the application being rejected or approved. This is the business process that we will use when looking at the General Concepts section of the training.

The second page shows another business process that we will use when reviewing our planning application. This business process will be designed so that it is reusable across multiple business processes. Advanced concepts will be introduced on this business process.

By the end of the General Concepts training you will have implemented the entire *Planning Application* business process, except for the review stage. By the end of the training you will have a complete business process which will make use of the generic review business process that you will also create.

Save a separate editable copy of the business process. As you go through the training change the style stages and connectors to the marked up style once they have been fully implemented.

Training:Architecture:Business Process Training Worksheets:BPM Package

Overview

The bpmmgr.bpm package provides several functions for getting information relevant to the current business process run. You can use these when running queries or APIs within your business process.

You may not understand where they are useful now but as you get an idea of the structure of the business process you should be able to see where they can be used. In the training there are examples of them being used and you will be required to use some of them when creating your business process definition.

Training:Architecture:Business Process Training Worksheets:Workbaskets

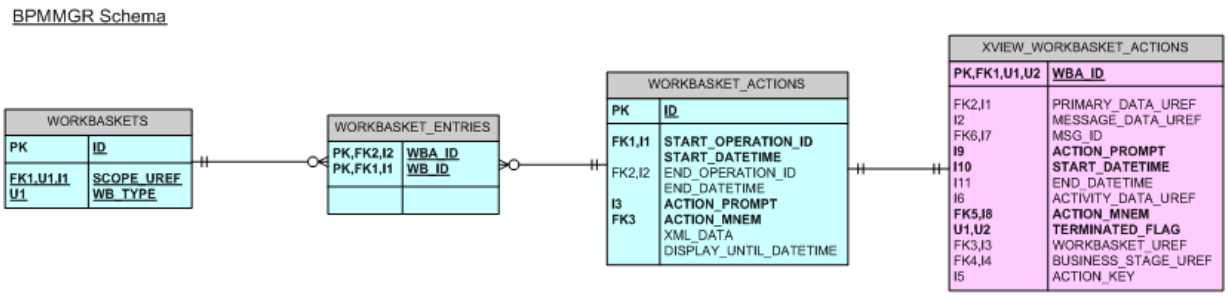
Introduction

FOX has a generic Workbasket sub-system which is intrinsically bound up with the BPM. This section will explain what it is and how it works.

Concepts

Fox has a generic Workbasket sub-system which application systems can utilise. As workflow is a repeating theme across most applications, the workbasket and work flow management is intrinsically bound up with the BPM. Note that the Workbasket Display has been implemented in Fox Module BPM001X. Manipulation of the workbasket contents however is strictly managed by the BPM. This ensures all workbasket changes are fully logged and auditable. Under no circumstances should an application program unit change workbasket data, other than through a BPM event call.

A workbasket is a logical container of work that can be shared between users (an audience). Workbaskets are currently scoped to individual users (WUA), system privileges (PRV), or team role instances (RRL) using Universal References (UREFs). Workbaskets are associated with workbasket actions which are items that appear in a user's workbasket which they can click on to carry out the work.



When planning to put work into an audience workbasket remember that if the audience changes, this will be reflected immediately in user workbaskets. E.g. by adding someone to a team they may get new workbasket actions out of the blue, or by removing someone, actions may disappear inexplicably (from the users viewpoint). In cases where this is unacceptable it is common practice to put work items into individual user workbaskets by scoping the workbasket to a WUA.

It is possible to create a workbasket action that is associated with multiple workbaskets at the same time. E.g. The action may be scoped to a workbasket for both a web_user_account and resource_role. This may be because the work item is always directed to the user that created it (the web_user_account), but also always copied to members of a given team role. When the workbasket is displayed to users, both the creator and team members will see the workbasket action. If the individual user just happens to also be a team member, they will still only be presented with one workbasket action, despite there being two reasons for relating it to this user. So it all makes good sense and works how you would expect.

The BPM001X (Workbasket) Fox Module merges logical workbaskets into a single workbasket before presenting it to users. So the end user simply just sees that they have workbasket actions, regardless of which workbasket scope delivered it to them.

As well as having a scope workbaskets also have a type. This indicates to the BPM001X (Workbasket) Fox Module where and when the workbasket contents should be displayed. Two generic workbasket types are currently used. A workbasket type of *WORK* will display its actions in the large workbasket list displayed in the centre of the screen. A workbasket type of *LHS* will display contents in the small menu in the left-hand-side of the display. Other generic types may be added during future development. As well as the generic types a developer can also define their own custom types. These can be any VARCHAR2(30) string and are defined in the BPD. Custom types are not displayed on the BPM001X module, instead you use these types in your own modules to display the actions for that type. This will be discussed later under the Panel Actions training sheet.

Training:Architecture:Business Process Training Worksheets:Business Context

Introduction

This section explains what the business context is, how it is used and why it is important.

Concepts

When a business process is started it creates a business routine. This represents the run of the business process. A business process may have many business routines in progress at the same time. The presents a problem with how we identify which routine we are working with when interacting with the business process. In order to distinguish between the different routines of the same business process (or in fact any routine of any business process) we use the business context.

The business context uniquely identifies a business process routine. This uniqueness is achieved by using UREFS, in the case below we are using just one UREF (PRIMARY_DATA). If we required more to make each routine unique then we can add other SUBJECT legs which have a PURPOSE of SECONDARY_DATA or maybe even TERTIARY_DATA (if we need to use 3 UREFS).

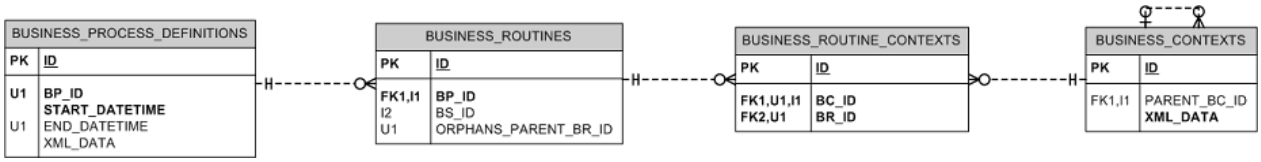
Business Context XML

```
<BUSINESS_CONTEXT>
  <CONTEXT>
    <SUBJECT>
      <REF_ID1>1MYUREF</REF_ID1>
      <USE>
        <PURPOSE>PRIMARY_DATA</PURPOSE>
```

```
          <PURPOSE>WORKBASKET</PURPOSE>
        </USE>
      </SUBJECT>
    </CONTEXT>
    <CONTEXT_NAME>PLANNING_APPLICATION_ROOT</CONTEXT_NAME>
  </BUSINESS_CONTEXT>
```

When interacting with the business process we look for business routines that match the PRIMARY_DATA in our context. If more than one routine is found we then narrow down our options by using the SECONDARY_DATA and finally, if necessary, the TERTIARY_DATA. The end result should be that we have found a single routine that matches the business process we are interacting with. This ensures that when a user carries out an action against their business process routine, their action does not have an effect on any other routines.

BPM Business Context Data Model



The root business context is constructed on create of the routine. During a business process extra data can be added to the context as and when it is needed for this routine. Child contexts may also be created to uniquely identify parts of a business routine. Therefore a single business routine may be associated with more than one context. You will see examples of this when we look at context switches later in the training.

Using the Business Context

As well as being used to uniquely identify the business routine, the UREFs in the business context will be UREFs applicable to your application. This means that you can access your application's ID from the business process to allow you to perform application specific functions.

The CONTEXT sub-element can be referenced using :{context} from the business process definition. You can also you the bpm.context_xml and bpm.set_context_xml methods from within the business process definition to get and set the context respectively.

A convenience PL/SQL database package UCTX provides a number of useful functions for processing CONTEXT data. These functions are worth knowing if you are processing a lot of CONTEXT data in your business process.

Training:Architecture:Business Process Training Worksheets:Delegations and Process Assignments

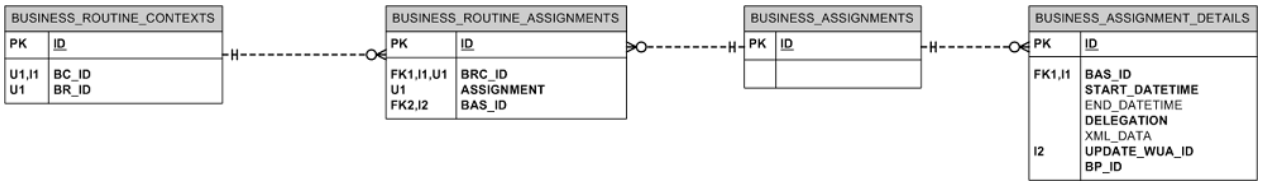
Introduction

This section will introduce you to delegation profiles and process assignments, and how they are used in the business process.

Concepts

Assignments define a person or groups of people that are used by the business process. The business process primarily uses them to determine what workbaskets an action should be assigned to, however they can also be used in other ways such as for sending an email to the users defined by the assignment.

BPM Assignment Data Model



An assignment is scoped to a business routine context. A process assignment is common across all business routine contexts. This means that if you have a business routine with more than one business context, each business context will be associated with the same process assignment. To use different assignments with each context we would use a contextual assignment. These are explained in more detail later in the training.

Business Assignment Detail XML

```
<ASSIGNMENT>
  <ASSIGNED_UREF_LIST>
    <UREF>1234WUA</UREF>
  </ASSIGNED_UREF_LIST>
  <DELEGATION_PROFILE>
    <DELEGATION>MY_DELEGATION_DL</DELEGATION>
    <DELEGATION_TYPE>CONFIG</DELEGATION_TYPE>
    <ASSIGNMENT_MANDATORY>false</ASSIGNMENT_MANDATORY>
    <MAKE_ASSIGNMENT_ON>NEVER</MAKE_ASSIGNMENT_ON>
    <ASSIGNEE_WHEN_UNASSIGNED_LIST>
      <RESOURCE_ROLE>
        <RES_TYPE>MY_RESOURCE_TYPE</RES_TYPE>
        <ROLE_NAME>MY_ROLE_NAME</ROLE_NAME>
        <NO_CONTEXT/>
      </RESOURCE_ROLE>
    </ASSIGNEE_WHEN_UNASSIGNED_LIST>
  </DELEGATION_PROFILE>
</ASSIGNMENT>
```

Process assignments are created at the start of the business process. Each assignment will hold information on valid assignments that can be made to it. This information is held in the delegation profile. The delegation profile will also contain information about when an assignment is automatically made (if ever) and whether or not an assignment can

be used by the business process if no assignments have be made to it.

An assignment can be made by scoping the assignment to one or more valid UREFs. In the above example the assignment has been scoped to the UREF *1234WUA*, which will be a web user account of a resource member in role *MY_ROLE_NAME*.

Automatically Making Assignments

A delegation can define if and when an assignment is automatically made in the *MAKE_ASSIGNMENT_ON* element. This can take the values *FIRST CREATE*, *FIRST USE* and *NEVER*.

1. *FIRST CREATE* makes an assignment to the users or roles specified in the delegation when the assignment is created at the start of the business process.
2. *FIRST USE* makes an assignment to the users or roles specified in the delegation when the assignment is first used in the business process.
3. *NEVER* means than an assignment will never automatically be made by the business process. This leaves you to control when an assignment is made and to who in your business process definition.

Delegation Types

A delegation also has a type. This type can be one of *CONFIG* or *POOL*.

1. The *CONFIG* type will perform no special actions and make assignments only based off what is specified in your business process definition.
2. The *POOL* type will make assignments automatically based on a "fairness" algorithm. It is generally used in conjunction with an automatic *FIRST USE* assignment to make an assignment to one of the user accounts, roles or privileges defined in the delegation on the first instance of the assignment being used in the business process.

Delegation Profile's Universal References (UREFs)

A delegation defines the workbaskets that an assignment can be made to in either an *ASSIGNEE_LIST* or an *ASSIGNEE_WHEN_UNASSIGNED_LIST*. These consist of XML markup or a SQL query that will return Universal References (UREFs) which can be instances of resource roles (RRL), web user accounts (WUA), and system privileges (PRV). Workbasket actions are defined in your business process and will make use these assignments to associate them with a workbasket.

If using *ASSIGNEE_WHEN_UNASSIGNED_LIST* then certain actions can be made available to the UREFs returned in the *ASSIGNEE_WHEN_UNASSIGNED_LIST* so long as no assignment has been made. Once an assignment has been made then only assignee actions will be available to the UREFs the assignment has been made to. E.g. If your delegation consists of a resource role in an *ASSIGNEE_WHEN_UNASSIGNED_LIST* then until a specific user in that role has been assigned then unassigned actions will be available to all users in the role. Once an assignment is made these actions will no longer be available to either the assigned user or any other user in the role. Only assignee action will be available to user the assignment has been made to. An additional point to note is that the *ASSIGNEE_WHEN_UNASSIGNED_LIST* cannot automatically make assignments, it must always by used in conjunction with *NEVER* in the *MAKE_ASSIGNMENT_ON* element.

The following methods are used to retrieve the UREFs for the delegation:

1. *CURRENT_USER* retrieves the web user account UREF (WUA) that initiated the current business process event.
2. *ROLE_USERS* retrieves the web user account UREFs (WUA) that exist for the specified resource role. Once an assignment has been made as this is scoped to a web user account rather than a resource role changes to the members in the resource role have no effect on the assignment. See additional note regarding contextual teams.

3. *RESOURCE_ROLE* retrieves the UREF for the specified resource role (RRL). Once an assignment is made as this is scoped to a resource role if a member is removed from the resource role they will lose the workbaskets scoped to that role and if they are added to the resource role they will gain the workbaskets scoped to the role. See additional note regarding contextual teams.
4. *SYSTEM_PRIV* retrieves the UREF for the specified system privilege (PRV). Once an assignment is made as this is scoped to a system privilege if the privilege is revoked access to the workbaskets scoped to that privilege will be lost and if the privilege is granted access to the workbaskets scoped to that privilege will be gained.
5. *QUERY_NAME* allows you to specify a query in the Business Process Definition query list.
6. *SQL* allow you to define a sql query.

Each row returned by the SQL query defined in the delegation or the Business Process Definition query list must return a Clob value in the format:

```
  <WORKBASKET>
    <SCOPE_UREF>WUA|RRL|PRV</SCOPE_UREF>
  </WORKBASKET>
```

For example:

```
SELECT
  XMLTYPE(XMLELEMENT("WORKBASKET"
  , XMLELEMENT("SCOPE_UREF", '100RRL')
  ).getclobval())
FROM dual
```

Note on contextual teams

If you specify a *RESOURCE_ROLE* or *ROLE_USERS* auto-query for a delegation, you may provide additional information if the resource is contextual (i.e. SCOPED_WITHIN = PARENT).

1. *CONTEXT_PURPOSE* is the *PURPOSE* of the UREF within the *business context* (PRIMARY_DATA|SECONDARY_DATA|TERTIARY_DATA)
2. *USAGE_PURPOSE* is the *PURPOSE* of the UREF as defined in the *RESOURCE_USAGES* table. Defaults to the value specified *CONTEXT_PURPOSE*.

These take the place of the <*NO_CONTEXT/*> element shown in the example markup below and ensure that the resource role instance used is the correct one for our team usage.

Assignments and Concurrent Access

We often use the business process to control concurrent access to a work item. For example, given an application we can use assignments to control who has access to edit that application at any one time. This can help to prevent lost updates or deadlocks which can occur if multiple users are allowed to edit the application at the same time. We do this by ensuring that when the assignment is made it is made to only a single web user account (WUA). So we may have a delegation defined with a resource role, this will control who assignments can be made to. When we actually make an assignment though we do not make it to the entire role, rather we make it to a single web user account associated to a member within the role. When doing this we must use the value *NEVER* for the *MAKE_ASSIGNMENT_ON* element. This is so that we can control who the assignment is made to in the Business Process Definition, rather than allowing the business process to automatically make the assignment in which case it would make it to the entire role.

BPD Markup

```
<ASSIGNMENTS>
  <DELEGATION_PROFILE_LIST>
    <DELEGATION_PROFILE>
      <DELEGATION>MY_DELEGATION_DL</DELEGATION>
      <DELEGATION_TYPE>CONFIG</DELEGATION_TYPE>
      <ASSIGNMENT_MANDATORY>false</ASSIGNMENT_MANDATORY>
      <MAKE_ASSIGNMENT_ON>NEVER</MAKE_ASSIGNMENT_ON>
      <ASSIGNEE_WHEN_UNASSIGNED_LIST>
        <RESOURCE_ROLE>
          <RES_TYPE>MY_RESOURCE_TYPE</RES_TYPE>
          <ROLE_NAME>MY_ROLE_NAME</ROLE_NAME>
          <NO_CONTEXT/>
        </RESOURCE_ROLE>
      </ASSIGNEE_WHEN_UNASSIGNED_LIST>
    </DELEGATION_PROFILE>
    <DELEGATION_PROFILE>
      <DELEGATION>MY_OTHER_DELEGATION_DL</DELEGATION>
      <DELEGATION_TYPE>POOL</DELEGATION_TYPE>
      <ASSIGNMENT_MANDATORY>true</ASSIGNMENT_MANDATORY>
      <MAKE_ASSIGNMENT_ON>FIRST USE</MAKE_ASSIGNMENT_ON>
      <ASSIGNEE_LIST>
        <ROLE_USERS>
          <RES_TYPE>MY_RESOURCE_TYPE</RES_TYPE>
          <ROLE_NAME>MY_OTHER_ROLE_NAME</ROLE_NAME>
          <NO_CONTEXT/>
        </ROLE_USERS>
      </ASSIGNEE_LIST>
    </DELEGATION_PROFILE>
  </DELEGATION_PROFILE_LIST>
  <PROCESS_ASSIGNMENT_LIST>
    <PROCESS_ASSIGNMENT>
      <ASSIGNMENT>MY_ASSIGNMENT</ASSIGNMENT>
      <ASSIGNMENT_TITLE>Sample resource role assignment</ASSIGNMENT_TITLE>
      <DELEGATION>MY_DELEGATION_DL</DELEGATION>
    </PROCESS_ASSIGNMENT>
    <PROCESS_ASSIGNMENT>
      <ASSIGNMENT>MY_OTHER_ASSIGNMENT</ASSIGNMENT>
      <ASSIGNMENT_TITLE>Sample pool role users assignment</ASSIGNMENT_TITLE>
      <DELEGATION>MY_OTHER_DELEGATION_DL</DELEGATION>
    </PROCESS_ASSIGNMENT>
  </PROCESS_ASSIGNMENT_LIST>
  <CONTEXTUAL_ASSIGNMENT_LIST/>
  <EXTERNAL_ASSIGNMENT_LIST/>
</ASSIGNMENTS>
```

This code will create two process assignments, each using a different delegation profile.

MY_ASSIGNMENT uses the *MY_DELEGATION_DL* delegation which has an *ASSIGNEE_WHEN_UNASSIGNED_LIST* returning an instance of a resource role. This delegation will not automatically make assignments allowing us to control when an assignment is made and to who in our BPD. This enables us to provide both unassigned actions to the entire role when no assignment has been made and assignee actions to the user we make an assignment to.

The second process assignment uses the *MY_OTHER_DELEGATION_DL* delegation which is of type *POOL* meaning that an assignment will be made to one of the users in the resource role on *FIRST USE* of the assignment. Note that when using a delegation of type *POOL* then it must be possible to make an assignment. E.g. using the above example if the instance of the resource role used (*MY_OTHER_ROLE_NAME*) has no members then attempting to make the *POOL* assignment would result in an application error. To ensure this does not happen you should only use resource roles for which it is mandatory to have one or more members with *POOL* delegation types.

Contextual assignments and external assignments have not been specified as these will be explained later in the training.

Exercises

Exercise 1

Create delegations meeting the following specification:

1. Applicant Delegation
    1. Uses the Applicant role in your team to provide a resource role for the delegation
    2. Requires an assignment to be made at all times
    3. Does not use unassigned actions
    4. Does not automatically make assignments
2. Viewer Delegation
    1. Uses the Viewer role in your team to provide a resource role for the delegation
    2. Does not require an assignment to be made at all times
    3. Uses unassigned actions
    4. Does not automatically make assignments
3. Case Manager Delegation
    1. Uses the Case Manager role in your team to provide a resource role for the delegation
    2. Does not require an assignment to be made at all times
    3. Uses unassigned actions
    4. Does not automatically make assignments
4. Document Manager Delegation
    1. Uses the Document Manager role in your team to provide resource role WUAs for the delegation
    2. Requires an assignment to be made at all times
    3. Does not use unassigned actions
    4. Automatically makes an assignment to ONE of the WUAs when the assignment is first used

Exercise 2

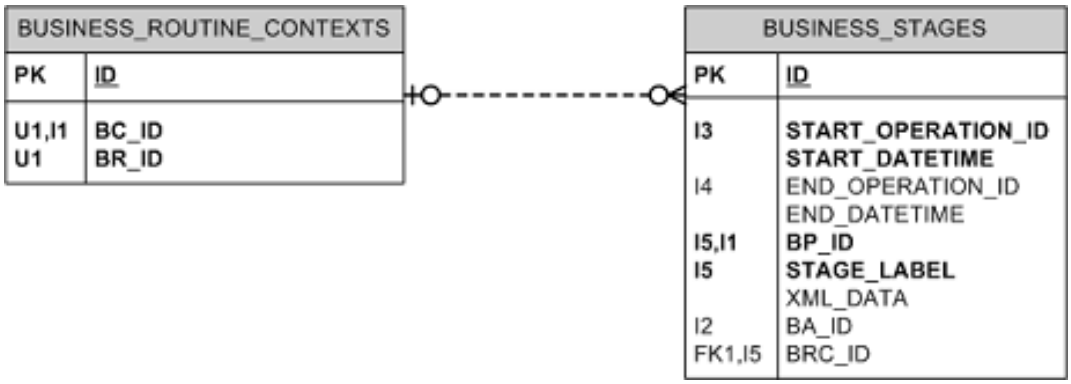
Link each of your delegations to a process assignment.

Training:Architecture:Business Process Training Worksheets:Business Stages

Overview

Business stages are used to control the flow of the workflow or to provide sets of actions to workbaskets enabling users to interact with the business process.

BPM Stages Data Model



Each stage will have a stage label which should match the label given to the stage on your workflow diagram. It will consist of a mnemonic common to all stages in your business process followed by a number. The stages should be numbered in sequence. We usually increment by 10 for each stage as this helps to keep the stages sequential even if we insert additional stages at a later point. The stage will also have a title which is used to identify the stage for reporting and tracking purposes.

As you move between stages on your business process they will be created and ended as required. A stage is scoped to a business routine context, which is used in conjunction with the stage label to uniquely identify the stage. You may have more than one instance of the same stage active on the same business routine as long as it is scoped to different business contexts within that routine allowing it to be uniquely identified by the BPM.

Stages can be marked up as an entry stage, exit stage, standard stage, conditional stage, sync stage or delay stage. We will go into each of these types in more detail later in the training.

Training:Architecture:Business Process Training Worksheets:Entry Stages

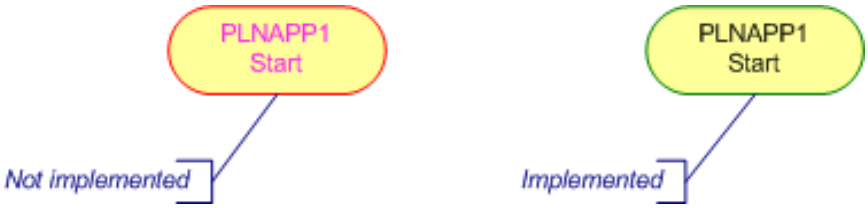
Introduction

This section will explain how entry stages are implemented in the business process definition.

Concepts

An entry stage is the start of the business process. There can only be one entry stage for a business process and you cannot provide a user with any actions on this stage. It requires a single transition, which is automatically run, to move to the business process on to the next stage. You can define tasks for the business process to undertake during this transition.

Workflow Markup



BPD Markup

```
<ENTRY_STAGE_LIST>
  <STAGE>
    <STAGE_LABEL>STAGE1</STAGE_LABEL>
    <STAGE_TITLE>My first entry stage</STAGE_TITLE>
  </STAGE>
</ENTRY_STAGE_LIST>
```

Exercises

1. Create a entry stage *PLNAPP1* for your business process.

Training:Architecture:Business Process Training Worksheets:Exit Stages

Introduction

This section will explain how exit stages are implemented in the business process definition.

Concepts

When a business process transitions it will create a new stage which once ended will in turn create another new stage. Exit stages are used to end business stages without moving them into a new stage. On moving into an exit stage the exit stage itself is then ended, thereby ending that path of the business process.

A business process is complete once all it stages are ended. If a business process has been split into several paths using a transition fork then each of these paths need to be ended before the routine will be complete.

We use the number 999 for exit stages (reducing by one for each extra stage required 998, 997, etc). In general all stages can transition to the same end stage. There are exceptions to this rule when working with subroutines which will be explained later in the training.

Workflow Markup

BPD Markup

```
<EXIT_STAGE_LIST>
  <STAGE>
    <STAGE_LABEL>STAGE999</STAGE_LABEL>
    <STAGE_TITLE>My first exit stage</STAGE_TITLE>
  </STAGE>
</EXIT_STAGE_LIST>
```

Exercises

1. Create exit stage *PLNAPP999* for your business process.

Training:Architecture:Business Process Training Worksheets:Standard Stages

Introduction

This section will outline how a standard stage is implemented in the business process definition and how it is used.

Concepts

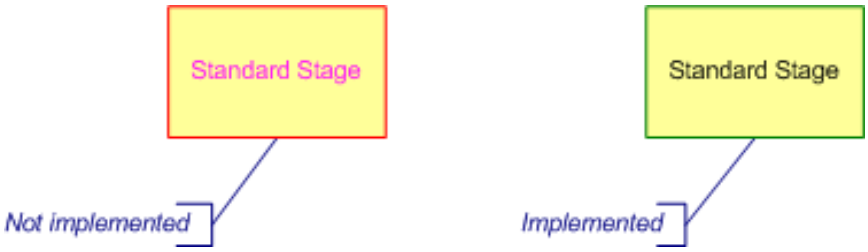
A standard stage allows users to interact with the business process. It does this by providing sets of actions, each of which are linked to workbaskets by associating them with one or more assignments.

On the main *WORK* workbasket each group of stage actions made available to the workbaskets the user can see are labelled with a description. This description is defined in the *ACTION_DESC_STAGE* element.

You can optionally define a default status for a standard stage. The status is primarily used by subroutines and we will explain how it is used in the Subroutines training worksheet. You define the default status of a stage in the *STATUS* element which takes either the values *ACTIVE* or *INACTIVE*. If not specified the default status is *ACTIVE*. Entry stages and conditional stages are always *ACTIVE* and exit stages are always *INACTIVE*.

Arguably the important part of a standard stage is the *ACTION_SET_LIST*. This list contains sets of actions each of which is associated with one or more workbaskets. These will be explained in more detail later in the training.

Workflow Markup



A standard stage is changed to the marked up format once all the action sets have been fully implemented.

BPD Markup

```
<STANDARD_STAGE_LIST>
  <STAGE>
    <STAGE_LABEL>STAGE10</STAGE_LABEL>
    <STAGE_TITLE>My Standard Stage 10</STAGE_TITLE>
    <ACTION_DESC_STAGE>Standard Stage 10</ACTION_DESC_STAGE>
    <ACTION_SET_LIST/> <!-- explained later -->
  </STAGE>
  <STAGE>
    <STAGE_LABEL>STAGE20</STAGE_LABEL>
    <STAGE_TITLE>My Standard Stage 20</STAGE_TITLE>
    <ACTION_DESC_STAGE>Standard Stage 20</ACTION_DESC_STAGE>
    <ACTION_SET_LIST/> <!-- explained later -->
  </STAGE>
</STANDARD_STAGE_LIST>
```

This code will define two standard stages. The markup for the stage actions hasn't been specified yet as we will cover this later in the training.

Exercises

1. Stub each standard stage on the planning application workflow.

Training:Architecture:Business Process Training Worksheets:Transitions

Introduction

Transitions define how the business process moves between the stages defined in its workflow. This section will explain how they are used.

Concepts

In order to move between stages the business process definition uses transitions. Transitions can be run by stage actions presented to the user, conditions on conditional stages, called explicitly in application code or in response to a signal, or run by the business process automatically for entry and sync stages.

Transition Basics

A basic transition will include a single before stage and a single after stage. On running the transition the following will take place:

1. The before stage will be ended, along with all its associated workbasket actions
2. The after stage will be created
3. Any assignments used the after stage that have a delegation profile that makes assignment on *FIRST USE* will be scoped to the UREFs returned by their delegation profile if they have not been used before.
4. Any workbaskets that need creating for assignments used on the after stage will be created.
5. The after stage's actions will be created and associated with the relevant workbaskets.

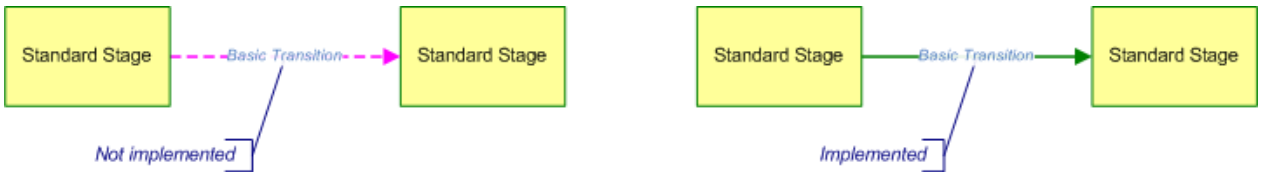
One or more tasks may also be performed. You can specify these to occur before, during or after the transition has taken place. Careful thought needs to be given as to when these should be performed to ensure that the correct information is available when it is needed. If the transition requires some work to be done whilst the before stage(s) still exists then use before transition, if the task relies on the existence of the after stage(s) then use after transition. In most instances carrying out work in transition will be suitable.

Common tasks performed by transitions include:

1. Making assignments
2. Running APIs
3. Sending emails to assignments

We will go into more detail on these tasks later in the training.

Workflow Markup



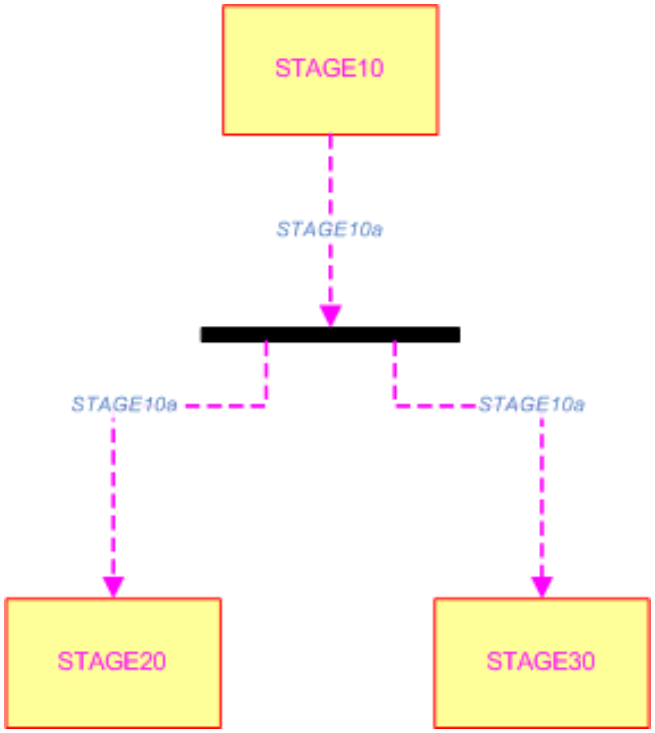
BPD Markup

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE10a</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE20</STAGE_LABEL>
    </AFTER>
  </MOVE>
</TRANSITION>
```

Transition Forks

A transition fork is used when a business process needs to go down two or more separate paths. This enables work to be done in parallel with transitions or actions taken in one part of the workflow not having any effect on the other part. A fork is implemented by providing a transition with more than one after stage, each after stage listed will be created when the transition is run. In order for the business routine to complete every stage must be ended. This means that at some point we must either end each path or join them back together.

Workflow Markup



BPD Markup

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE10a</TRANSITION_LABEL>
```

```
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE20</STAGE_LABEL>
    </AFTER>
    <AFTER>
      <STAGE_LABEL>STAGE30</STAGE_LABEL>
    </AFTER>
  </MOVE>
</TRANSITION>
```

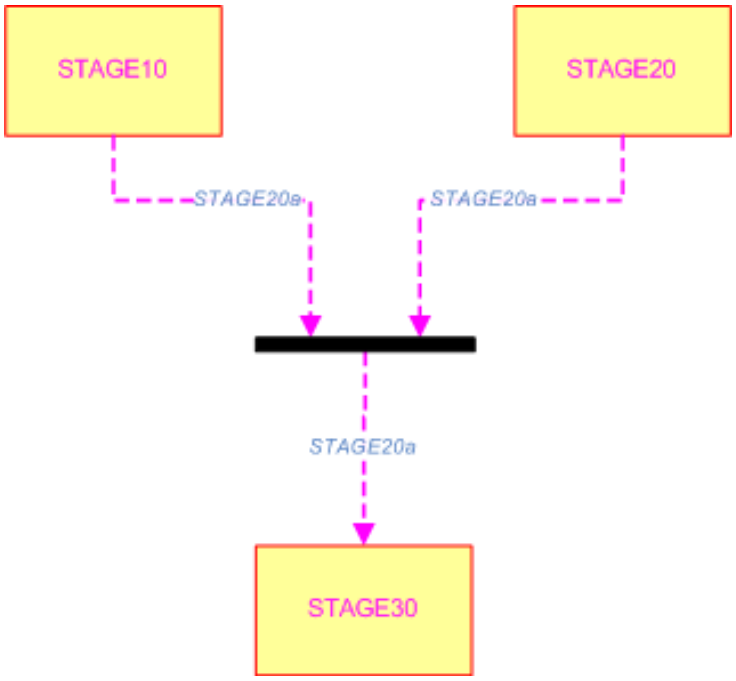
Transition Joins

If we have split a business process into separate paths we can join these paths back together by using a transition join. A transition join takes multiple before stages and a single after stage. The transition that runs the transition join is generally named using the highest numbered stage from its before stages excluding any stages that will never invoke the transition. On being invoked the transition will end each before stage listed and create the after stage specified.

By default the transition can be invoked by any of the before stages and every before stage has to exist, otherwise an exception will be raised and the transition will fail. You can override this behaviour by specifying the following optional attributes:

1. *INVOKER_SIBLING* defines whether the stage needs to exist as a sibling of the before stage that invoked the transition. This takes the values *MUST-EXIST* (default) which will raise an exception if the stage does not exist when the transition is invoked by another stage and *MAY-EXIST* which allows it to not exist.
2. *INVOKER_THIS* defines whether the before stage that raised the transition needs to exist. This takes the values *MUST-EXIST* (default) which will raise an exception if the stage does not exist and *NOT-INVOKED* which is for before stages that will not invoke this transition.

Workflow Markup



BPD Markup

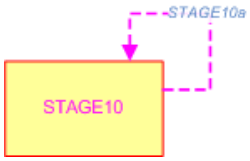
```
<TRANSITION>
  <TRANSITION_LABEL>STAGE20a</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
      <INVOKER_THIS>NOT-INVOKED</INVOKER_THIS>
      <INVOKER_SIBLING>MAY-EXIST</INVOKER_SIBLING>
    </BEFORE>
    <BEFORE>
      <STAGE_LABEL>STAGE20</STAGE_LABEL>
      <INVOKER_THIS>MUST-EXIST<INVOKER_THIS>
      <INVOKER_SIBLING>MUST-EXIST</INVOKER_SIBLING>
    </BEFORE>
    <BEFORE>
      <STAGE_LABEL>STAGE30</STAGE_LABEL>
      <INVOKER_THIS>NOT-INVOKED</INVOKER_THIS>
      <INVOKER_SIBLING>MUST-EXIST</INVOKER_SIBLING>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE40</STAGE_LABEL>
    </AFTER>
  </MOVE>
</TRANSITION>
```

In the above example *STAGE10* may not exist when the transition is invoked and will never be the stage that invokes the transition. *STAGE20* may invoke the transition and must exist when the transition is invoked. *STAGE30* must also exist when the transition is invoked but will never invoke the transition itself. We have labelled the transition *STAGE20a* as that is the only stage that can invoke the transition.

Self Transitions

There are times when you may want to refresh a stage. This is most commonly used when changing between unassigned and assigned assignments and to create their associated workbasket actions. In order to do this you can transition a stage back onto itself. An after stage can include the optional element *EXISTS* which defaults to *MUST-NOT-EXIST*. If you want to transition a stage back onto itself you must set the value of this element to *MAY-EXIST*.

Workflow Markup



BPD Markup

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE10a</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
```

```
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
      <EXISTS>MAY-EXIST</EXISTS>
    </AFTER>
  </MOVE>
</TRANSITION>
```

Exercises

1. Add the transitions for stages PLNAPP1 and PLNAPP10 to your business process definition. For now do not worry about the tasks that these stages perform when they are run.

NOTE: Until you have implemented all the tasks performed by a transition do not mark it up as complete on your workflow diagram.

Training:Architecture:Business Process Training Worksheets:Workbasket Actions

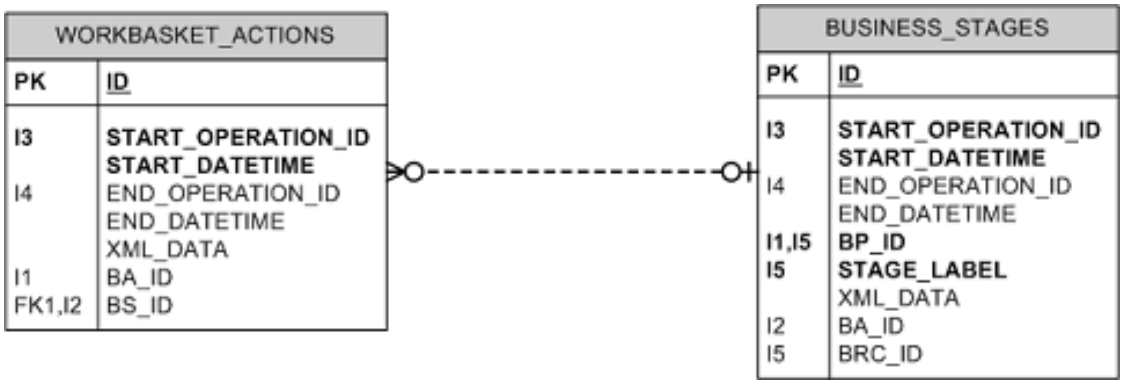
Introduction

This section explains how workbasket actions are made available to users and what they can do.

Concepts

A standard stage in a business process will provide one or more sets of actions which it makes available to certain workbaskets. These actions are used to enable users to interact with the business process to perform tasks such as running FOX source code or transitioning the business process.

BPM Stage Actions Data Model



Action Sets

An action set is uniquely identified for a stage by its mnemonic. This is generally a sequence of numbers or roman numerals.

Workbasket Assignments

Each action set contains a *WORKBASKET_ASSIGNMENT_LIST* which contains one or more workbasket assignments. Each workbasket assignment will be associated with an assignment defined in your business process and a workbasket type. The assignment defines the workbaskets that the action set is associated with. On entering a stage workbaskets will be created for each workbasket assignment in an action set if it does not already exist. The action set will then be created and associated with these workbaskets. If you want your actions to appear on the central workbasket in *BPM001X* the workbasket type should be set to *WORK*.

You can also define an optional element in your workbasket assignment called *ASSIGNMENT_GROUP*, which takes one of two values:

1. *ASSIGNEE-WHEN-ASSIGNED* workbasket assignments will only be created and associated to action sets if the assignment has been scoped to one or more UREFs. If no assignment has been made then this action set will not be available to the workbasket assignment.
2. *ASSIGNEE-WHEN-UNASSIGNED* can be used if your delegation profile uses an *ASSIGNEE-WHEN-UNASSIGNED_LIST*. Workbasket assignments with this markup will only be created and associated to action sets if the assignment has **not** been scoped to any UREFs. If no assignment has been made then this action set will be available for all the UREFs returned by the *ASSIGNEE-WHEN-UNASSIGNED_LIST* in the delegation profile used by the assignment.

If not specified the default *ASSIGNMENT_GROUP* is *ASSIGNEE-WHEN-ASSIGNED*.

Actions

An action set will have one or more associated actions. These are the actions the users will see in their workbaskets and which they will be able to use to interact with the business process.

An action has an action prompt which will be displayed on the action and an action order which defines the order in which actions are displayed. The action order must be unique across all actions for the stage.

Action Source Code

You can define action source code for the action to run. This is FOX module code that is run before any transitions take place. It is often used for calling into a FOX module from the central workbasket, or calling an action in your module to carry out validation or to create/update data required for a transition that cannot be done in a before transition API.

When performing validation a common pattern is to use an alert to inform the user if the validation failed and follow this up by stopping the the transaction from proceeding by throwing an *ACTIONBREAK*. This will halt the BPM and prevent any transitions from being run.

```
<fm:alert message="You cannot do this because of xxxxx"/>
<fm:throw code="ACTIONBREAK" message="Validation failed."/>
```

Transition Label

If an action defines a transition label then the targeted transition will run when the action is clicked. The transition will run after the action source code and any action data (explained later) that is present.

After Transition Source Code

After transition source code can only be present if the action has a transition label. This element defines action code to be run after the transition has completed.

BPD Markup

```
<ACTION_SET_LIST>
  <ACTION_SET>
    <ACTION_SET_MNEM>1</ACTION_SET_MNEM>
    <WORKBASKET_ASSIGNMENT_LIST>
      <WORKBASKET_ASSIGNMENT>
        <ASSIGNMENT>EDITOR</ASSIGNMENT>
        <WORKBASKET>WORK</WORKBASKET>
      </WORKBASKET_ASSIGNMENT>
    </WORKBASKET_ASSIGNMENT_LIST>
    <ACTION_LIST>
      <ACTION>
        <ACTION_PROMPT>Edit My Stuff</ACTION_PROMPT>
        <ACTION_ORDER>10</ACTION_ORDER>
        <ACTION_SOURCE_CODE>
          <assign initTarget=":{temp}/MY_ID" expr="substring-before(:{context}/SUBJECT[USE/PURPOSE='PRIMARY_DATA']/REF_ID1, 'MY_UREF_TYPE')"/>
          <call-module module="MY_MODULE" theme="edit" type="modal" params=":{temp}/*"/>
        </ACTION_SOURCE_CODE>
      </ACTION>
      <ACTION>
        <ACTION_PROMPT>Submit My Stuff</ACTION_PROMPT>
        <ACTION_ORDER>20</ACTION_ORDER>
        <TRANSITION_LABEL>STAGE10a</TRANSITION_LABEL>
        <AFTER_TRANSITION_SOURCE_CODE>
          <alert message="Submitted successfully."/>
        </AFTER_TRANSITION_SOURCE_CODE>
      </ACTION>
    </ACTION_LIST>
  </ACTION_SET>
  <ACTION_SET>
    <ACTION_SET_MNEM>2</ACTION_SET_MNEM>
    <WORKBASKET_ASSIGNMENT_LIST>
      <WORKBASKET_ASSIGNMENT>
        <ASSIGNMENT>VIEWER</ASSIGNMENT>
        <ASSIGNMENT_GROUP>ASSIGNEE-WHEN-UNASSIGNED</ASSIGNMENT_GROUP>
        <WORKBASKET>WORK</WORKBASKET>
      </WORKBASKET_ASSIGNMENT>
    </WORKBASKET_ASSIGNMENT_LIST>
```

```
    <ACTION_LIST>
      <ACTION>
        <ACTION_PROMPT>View My Stuff</ACTION_PROMPT>
        <ACTION_ORDER>30</ACTION_ORDER>
        <ACTION_SOURCE_CODE>
          <assign initTarget=":{temp}/MY_ID" expr="substring-before(:{context}/SUBJECT[USE/PURPOSE='PRIMARY_DATA']/REF_ID1, 'MY_UREF_TYPE')"/>
          <call-module module="MY_MODULE" theme="view" type="modal" params=":{temp}/*"/>
        </ACTION_SOURCE_CODE>
      </ACTION>
    </ACTION_LIST>
  </ACTION_SET>
</ACTION_SET_LIST>
```

The above code shows two action sets. The actions in the first action set will be available to to the UREFs the *EDITOR* assignment is scoped to. This action set contains two actions.

1. The first action gets the ID for your application from the business context's primary data UREF and passes this in as a parameter to your module call.
2. The second action calls a transition to move the workflow on to the next stage and informs the user with a alert once this has been done.

The actions in the second action set will be available to all the UREFs associated with the delegation profile the *VIEWER* assignment uses (so long as the a viewer has not been assigned). This action set has a single action which again gets the ID for your application before calling into the view theme on your module.

Exercises

Entering Your Module

Create an action set under stage *PLNAPP10* with the following specification:

1. Is available to your applicant assignment
2. Is only available when your applicant assignment has been given a scope (is assigned)
3. Is available on the central workbasket
4. Has a single action which will call your planning application module under the *edit* theme with the parameters required by your module (i.e. your ID)

Unassigned Workbasket Action

Create an action set under stage *PLNAPP10* with the following specification:

1. Is available to your viewer assignment
2. Is only available when your applicant assignment has **not** been given a scope (is not assigned)
3. Is available on the central workbasket
4. Has a single action which will call your planning application module under the *view* theme with the parameters required by your module (i.e. your ID)

Using Transitions

Create an action set under stage *PLNAPP10* with the following specification:

1. Is available to your applicant assignment
2. Is only available when your applicant assignment has been given a scope (is assigned)
3. Is available on the *PANEL* workbasket (this will be explained later)
4. Has a single action that:
    1. Runs the transition *PLNAPP10c*
    2. Runs after transition source code to inform the user their Planning Application has been cancelled and exit the module

Using Action Source Code

Create an action set under stage *PLNAPP10* with the following specification:

1. Is available to your applicant assignment
2. Is only available when your applicant assignment has been given a scope (is assigned)
3. Is available on the *PANEL* workbasket (this will be explained later)
4. Has a single action that:
    1. Calls an action (defined in your FOX module) to validate the application and terminate if the validation fails.
    2. Runs the transition *PLNAPP10a*
    3. Runs after transition source code to inform the user their Planning Application has been submitted and exit the module

Training:Architecture:Business Process Training Worksheets:Action Data

Introduction

This section will introduce you to the *ACTION_DATA* element in an *ACTION_SET*.

Concepts

Action data is defined for an action set, so it is the same for all actions in the action set. This means you need to be sure that the action data is relevant to all the actions in the action set, if it is not you may need to consider moving those actions where it is not relevant to a different action set.

Action data is used for four main purposes which are outlined below.

Confirm Dialogs

Confirm dialogs provide a means of pausing the business process when an action is clicked to provide a confirm dialog to the user that triggered the action. The confirm dialog is displayed before any action code or transitions are run. If the user selects *OK* on the confirm dialog the business process will continue to run the action, if the user selects *Cancel* the action will terminate.

BPD Markup

```
<ACTION_SET>
  <WORKBASKET_ASSIGNMENT_LIST>
    ...
  </WORKBASKET_ASSIGNMENT_LIST>
  <ACTION_LIST>
    ...
  </ACTION_LIST>
  <ACTION_DATA>
    <ACTION_CONFIRM>Are you sure you would like to delete this case?</ACTION_CONFIRM>
  </ACTION_DATA>
</ACTION_SET>
```

Background workbaskets

On the central workbasket you can define filters to change which actions are visible at any one time. When defining actions in your business process definition they default to the *For Attention* filter. This is the default view for the central workbasket.

In the business process you can override the default filter for the actions in the action set. You do this by providing a value for the *ACTION_CATEGORY* element under *ACTION_DATA*. When the actions in this action set are created they will use the value defined by action category as their filter. This means they will not be viewable in the default *For Attention* workbasket. The user will have to change their filter tag to the one set by the action

Action category should generally only be used for actions that are not very important to the user or do not require immediate action. It can be very easy for users to forget about items in their background workbaskets, thereby leaving them incomplete for long periods of time. Also some users may not fully understand the filters on the workbasket leading them to believe their case has been 'lost' as they cannot find the actions for it.

BPD Markup

```
<ACTION_SET>
  <WORKBASKET_ASSIGNMENT_LIST>
    ...
  </WORKBASKET_ASSIGNMENT_LIST>
  <ACTION_LIST>
    ...
  </ACTION_LIST>
  <ACTION_DATA>
    <ACTION_CATEGORY>My Filter</ACTION_CATEGORY>
  </ACTION_DATA>
</ACTION_SET>
```

Action Descriptions

All action sets are described by the stage. If you want to add a further description for a particular action set you can by using the *ACTION_DESC_ACTION* element under *ACTION_DATA*. This will append your action set's action desc to the stage's action desc separated by a comma.

BPD Markup

```
<ACTION_SET>
  <WORKBASKET_ASSIGNMENT_LIST>
    ...
  </WORKBASKET_ASSIGNMENT_LIST>
  <ACTION_LIST>
    ...
  </ACTION_LIST>
  <ACTION_DATA>
    <ACTION_DESC_ACTION>Description</ACTION_DESC_ACTION>
  </ACTION_DATA>
</ACTION_SET>
```

Pause Panels

On clicking an action in an action set with a pause panel the pause panel will be activated first and pause the flow of execution allowing the user to provide information to the business process before proceeding. The main use of a pause panel is for reassigning a case in order to allow the user to select the web user account the case will be reassigned to. This is explained in more detail later in the training under Transition Assignments.

Exercises

Add Action Confirms

Add an action confirm dialog for the action sets with your Submit and Cancel Application actions asking the user to confirm their action.

Set the Action Category

Add an action category for your view application action to move it into a background workbasket.

Add Extra Description

Add an action description to your view application action to specify that it is 'In Progress'.

Training:Architecture:Business Process Training Worksheets:Panel Actions

Introduction

This section will explain what panel actions are and how they are used.

Concepts

As well as workbasket actions which will appear in the central workbasket, the actions in an action set can also be defined as being panel actions. This allows you to group sets of actions on a stage to include in your application modules.

A panel action is defined in exactly the same way as a central workbasket action, except instead of using a workbasket type of *WORK* or *LHS* you use the workbasket type *PANEL*. In fact the workbasket type you use can be any VARCHAR2(30) string. This allows you to have different groups of actions on the same stage which may then be used on different screens or set-outs on the same screen.

Including Panel Actions In Your Module

In order to add panel actions to your module you need to know two things:

1. The workbasket type of the actions you want to display.
2. The business stage ID of the stage the panel actions exists on. When a workbasket action is clicked the action XML is written to the environment DOM. The data from the environment DOM is passed forward on module calls, so when triggering a workbasket action from the central workbasket that calls into your module (either directly or indirectly via a chain of module calls) then then that workbasket action's XML will be available. This XML contains the business stage ID of the workbasket action that was clicked in the element *BS_ID* as shown below.

```
<WORKBASKET_ACTION>
  <ACTION_MNEM>i.STAGE10.MY_BPD</ACTION_MNEM>
  <CONTEXT>
    <SUBJECT>
      <REF_ID1>1MYUREF</REF_ID1>
      <USE>
        <PURPOSE>PRIMARY_DATA</PURPOSE>
        <PURPOSE>WORKBASKET</PURPOSE>
      </USE>
    </SUBJECT>
  </CONTEXT>
  <CONTEXT_NAME>MY_BPD_ROOT</CONTEXT_NAME>
  <STAGE_LABEL>STAGE10</STAGE_LABEL>
  <STAGE_TITLE>My First Stage</STAGE_TITLE>
  <STAGE_STATUS>ACTIVE</STAGE_STATUS>
  <BS_ID>1128994</BS_ID>
  <TALLY_LIST/>
</WORKBASKET_ACTION>
```

Unless you transition to another stage this will be the business stage ID that you will need to display your panel actions.

Using this information you are now ready to add panel actions on to your module. To do this you need to carry out the following steps:

- Library in FOX module *BPM002X* which has several actions available for displaying and working with panel actions.
- Add an element to your schema under theme named *WB_PANEL* of type *WB_PANEL_TYPE*. This defines the XML structure the data on the panel actions will take.

```
<xs:element name="WB_PANEL" type="WB_PANEL_TYPE"/>
```

- Add an action that populates the WB_PANEL element with your panel actions. To do you need to init a *:{theme}/WB_PANEL/WB_PANEL_LIST* with the workbasket type of your panel actions and *:{theme}/WB_PANEL/BS_ID* with the business stage your panel actions are on. Then set a context of *panel* targeting *:{theme}/WB_PANEL* and call the *BPM002X* action *BPM002X-action-refresh-WB-panel* which will retrieve a list of panel actions matching your stage, workbasket type and workbaskets your user is associated with. An example of this is shown below:

```
<fm:action name="action-refresh-panel-actions">
  <fm:do>
    <!-- set workbasket type to PANEL -->
    <fm:assign initTarget=":{theme}/WB_PANEL/WB_PANEL_LIST" textValue="PANEL"/>
    <!-- set BS_ID to stage ID of last workbasket action clicked -->
    <fm:assign initTarget=":{theme}/WB_PANEL/BS_ID" expr=":{env}/WORKBASKET_ACTION/BS_ID"/>
    <fm:context-set scope="state" name="panel" xpath=":{theme}/WB_PANEL"/>
    <!-- query out panel actions -->
    <fm:call action="BPM002X-action-refresh-WB-panel"/>
  </fm:do>
</fm:action>
```

- Call this action from each entry theme that needs to use your panel actions to query your panel action into the theme DOM.
- Include the buffer *BPM002X-wb-panel-buttons-across* in each state where you want your panel actions to be visible. This buffer sets out the panel actions that were retrieved by the BPM002X-action-refresh-WB-panel action.

```
<fm:include name="BPM002X-wb-panel-buttons-across" attach=":{theme}/WB_PANEL"/>
```

Note: In order to get your module to validate you will need to import BPM002X which is where the *WB_PANEL_TYPE* is defined.

```
<xs:import schemaLocation="C:\LOCATION_OF_TRAINING_DIRECTORY\BPM002X.xml"/>
```

Exercises

Display your panel actions in the state associated with your *edit* theme on your planning application module.

Training:Architecture:Business Process Training Worksheets:Make Assignment

Introduction

One of the tasks you can carry out in a transition is making an assignment. This section will explain how this is carried out and when you may want to do it.

Concepts

As explained in the Delegations and Process Assignments training sheet a business process will have one or more assignments which can be scoped to all of the UREFs provided in the assignment's delegation profile on the first use of the assignment or on the assignment being created. Alternatively, we can choose to scope the assignment to specific UREFs in the delegation profile at specific points in the business process.

There are a number of common scenarios in which you may want to scope an assignment to a UREF. These are outlined below:

Moving To A New Stage

When moving to a new stage on which the assignment is used, if an assignment has not been made yet you may at this point decide to scope the assignment to one (or more) of the UREFs provided in the assignment's delegation profile. This is different from the *FIRST USE* method in the delegation profile which will scope the assignment to all the UREFs defined in the delegation profile. Additionally, if the delegation profile uses an instance of a resource role or a system privilege rather than scoping the assignment to these UREFs the UREF we scope the assignment to can be that of any web user account that is associated with that role or privilege.

This type of assignment is commonly used when the web user account that invoked the transition is a valid web user account for the assignment. In this case we can scope the assignment to the user that invoked the transition who must be a valid user for the *EDITOR* assignment. This means that when the new stage is created they will have access to all the assignee actions for that assignment.

BPD Markup

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE1a</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE1</STAGE_LABEL>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
    </AFTER>
  </MOVE>
  <IN_TRANSITION>
    <ASSIGNMENT_ASSIGN_LIST>
      <ASSIGNMENT>EDITOR</ASSIGNMENT>
      <CURRENT_USER/>
    </ASSIGNMENT_ASSIGN_LIST>
```

```
  </IN_TRANSITION>
</TRANSITION>
```

In this example we have moved from *STAGE1* to *STAGE10*, scoping the *EDITOR* assignment to the current user which will be the UREF of the web user account that invoked the transition.

Taking Ownership

At times we may pass a case forward to a stage which requires work to be done by users in another assignment. We may not know which users associated with the assignment should carry out the work so we cannot scope the assignment to a particular web user account ourselves. This means that we need to provide all the users associated with the assignment a way in which to scope the assignment to themselves, allowing the correct user to take control.

In order to do this we need to use unassigned actions, so the delegation profile needs to be setup to allow this by using the *ASSIGNEE_WHEN_UNASSIGNED_LIST*. We can then provide an unassigned action to all the users associated with the assignment which when clicked will scope the assignment to themselves. We can provide this action by defining an action set with a workbasket assignment for our assignment with an *ASSIGNMENT_GROUP* of *ASSIGNEE-WHEN-UNASSIGNED*. The actions in this action set will then be available to all the UREFs associated with the assignment until an assignment is made.

In order to scope the assignment to themselves we provide an unassigned action that will invoke a transition which makes an assignment in transition. This transition will carry out the following actions:

1. End the workbasket actions associated with the stage
2. Run the transition tasks (in this case we will make an assignment in the same way as when moving to a new stage)
3. Reassociate the stages workbasket actions. As the assignment now has a scope the unassigned actions will not get re-created, only assigned actions for the assignment will be created.

BPD Markup

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE10a</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
      <EXISTS>MAY-EXIST</EXISTS>
    </AFTER>
  </MOVE>
  <IN_TRANSITION>
    <ASSIGNMENT_ASSIGN_LIST>
      <ASSIGNMENT>EDITOR</ASSIGNMENT>
      <CURRENT_USER/>
    </ASSIGNMENT_ASSIGN_LIST>
  </IN_TRANSITION>
</TRANSITION>
```

Reassigning

At times we may want to change the scope of an assignment from one UREF to another. E.g. Say we have a user that is drafting an application and this requires another user to carry out work on the same application. Rather than scoping the assignment to both users, allowing them both to edit the application and therefore introducing potential concurrency issues, we can scope the assignment to one of the users and when required they can request to reassign the application, selecting another user associated with the assignment to scope the assignment to.

To do this we use a pause panel. On clicking an action with a pause panel the pause panel will be activated first and pause the flow of execution allowing the user to provide information to the business process before proceeding. In this instance the information we will be providing to the business process is the web user account of the user we want to reassign the case to. An action using a pause panel must be implemented as a panel action, it does not work when placed on the central workbasket.

To use a pause panel no changes need to be made to your FOX module beyond displaying the panel action as all the work is carried out in *BPM002X*.

A pause panel is implemented in your business process using the *PANEL_ACTION_PAUSE_LIST* under the *ACTION_DATA* element. To control who the business process can be reassigned to, you provide the pause panel with the assignment from your business process that is being reassigned.

The action set should have an action that runs a transition that will refresh the stage by using the same stage for the before and after stages in the transition. This is similar to taking ownership except that you do not make an assignment in transition as the assignee is provided by the pause panel.

BPD markup

```
<ACTION_SET>
  <ACTION_SET_MNEM>1</ACTION_SET_MNEM>
  <WORKBASKET_ASSIGNMENT_LIST>
    <WORKBASKET_ASSIGNMENT>
      <ASSIGNMENT>EDITOR</ASSIGNMENT>
      <WORKBASKET>PANEL</WORKBASKET>
    </WORKBASKET_ASSIGNMENT>
  </WORKBASKET_ASSIGNMENT_LIST>
  <ACTION_LIST>
    <ACTION>
      <ACTION_PROMPT>Reassign Case</ACTION_PROMPT>
      <ACTION_ORDER>10</ACTION_ORDER>
      <TRANSITION_LABEL>STAGE10a</TRANSITION_LABEL>
      <AFTER_TRANSITION_SOURCE_CODE>
        <alert message="Your case has been reassigned."/>
        <exit-module/>
      </AFTER_TRANSITION_SOURCE_CODE>
    </ACTION>
  </ACTION_LIST>
  <ACTION_DATA>
    <PANEL_ACTION_PAUSE_LIST>
      <PANEL_ACTION_PAUSE>
        <SELECT_ASSIGNEE_ASSIGNMENT>EDITOR</SELECT_ASSIGNEE_ASSIGNMENT>
      </PANEL_ACTION_PAUSE>
   </PANEL_ACTION_PAUSE_LIST>
```

```
  </ACTION_DATA>
</ACTION_SET>
```

Exercises

Assignment When Moving Between Stages

Scope the *APPLICANT* assignment to the user that started the business process on transition *PLNAPP1a*.

Taking Ownership

Add an action set to stage *PLNAPP40* with the following specification:

1. Is available to your case manager assignment
2. Is only available when your applicant assignment has **not** been given a scope (is unassigned)
3. Is available on the central workbasket
4. Has a single action which when clicked will scope the case manager assignment to the user that triggered the action

Reassigning

Add an action set to stage *PLNAPP10* with the following specification:

1. Is available to your applicant assignment
2. Is only available when your applicant assignment has been given a scope (is assigned)
3. Is available on the *PANEL* workbasket
4. Has a single action which will make a reassignment to another web user account for your applicant assignment
5. After making the assignment provides feedback to the user and exit the module

Training:Architecture:Business Process Training Worksheets:Run API

Introduction

A common task to perform in a transition is to run an API specified in your business process. This section will cover how to specify API's, where they can be run from and how to run them in a transition.

Concepts

You will often have to perform complex application logic when moving between stages in your business process. To enable you to do this a transition allows you to run an API which can contain either a SQL query, PL/SQL anonymous block or a number of conditional clauses. These APIs are defined in the business process definition's *QUERY_DEFINITION_LIST*.

Query Definition List

The query definition list can contain one or more named query definitions. These query definitions can be used throughout your business process when running transitions, to define the UREF's for a delegation profile, from condition clauses, action overlays and on stage events.

A query definition has a name and a SQL block or conditional elements (*WHEN|AND|OR|NOT|NO*).

The SQL block can consist of a SQL statement or PL/SQL anonymous block. If performing application specific tasks these will be held in the application's package(s) and the API will make a call to the package in order to carry out the task. An example of this is shown below.

```
  <QUERY_DEFINITION>
    <QUERY_NAME>Do Stuff</QUERY_NAME>
    <SQL>
DECLARE
  l_my_id NUMBER := REPLACE(uctx.getrefid1(bpm.context_xml,
'PRIMARY_DATA'), 'MY_UREF');
BEGIN
  my_schema.my_package.do_stuff(
    p_id => l_my_id
  );
END;
    </SQL>
  </QUERY_DEFINITION>
```

The *WHEN* element defines a conditional statement that must return true or false. It works similarly to the *WHERE* clause in a SQL statement. You can make calls to functions in packages or other query definitions comparing the values returned to return true or false.

```
  <QUERY_DEFINITION>
    <QUERY_NAME>Check Stuff</QUERY_NAME>

    <WHEN>my_schema.my_package.count_some_stuff(REPLACE(uctx.getrefid1(bpm.context_xml,
```

```
'PRIMARY_DATA'), 'MY_UREF'))>0</WHEN>
 </QUERY_DEFINITION>
```

The *AND|OR|NOT|NO* elements perform use conditional logic to return a single true or false value. You can use them to call query definitions that return true or false (are defined with the *WHEN* element)

```
<QUERY_DEFINITION>
  <QUERY_NAME>Check Some More Stuff</QUERY_NAME>
  <AND>
    <QUERY_NAME>Check Stuff</QUERY_NAME>
    <NOT>
      <QUERY_NAME>Check Other Stuff</QUERY_NAME>
    </NOT>
  </AND>
</QUERY_DEFINITION>
```

Running an API in Transition

You can run one or more APIs IN|BEFORE|AFTER transition. You do this by specifying the name of the query definition(s) you want to run.

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE10a</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE20</STAGE_LABEL>
    </AFTER>
  </MOVE>
  <IN_TRANSITION>
    <API_LIST>
      <QUERY_NAME>Do Stuff</QUERY_NAME>
      <QUERY_NAME>Do Some More Stuff</QUERY_NAME>
    </API_LIST>
  </IN_TRANSITION>
<TRANSITION>
```

Exercises

Create Query Definitions

Create two query definitions. Each definition should call the procedure trainingmgr.xx_plan_app.update_application_status(p_application_id, p_new_status) in your package. One should provide a status *SUBMITTED* the other should provide the status *CANCELLED* for the p_new_status parameter.

Use Query Definitions in Transitions

1. Transition *PLNAPP10a* should call the query definition that updates your application to submitted.
2. Transition *PLNAPP10c* should call the query definition that updates your application to cancelled.

Training:Architecture:Business Process Training Worksheets:LHS Workbasket Actions

Introduction

This section will show you how to make static left hand side workbasket actions available.

Concepts

For most users the workbasket on *BPM001X* is used as the portal dashboard from where they can jump to ongoing business processes which require some form of action to be taken by them. On this screen you can also provide users with actions that appear on the left hand side of the screen which are generally used to perform non-business process related activities. For these actions you will want them to be visible at all times so long as the user has the relevant permissions.

In order to get these actions to appear we use a different type of business process that defines static events, operations and actions. We will generally create a single business process for our application that will define all the static LHS actions that are to be made available for it. To identify that the business process is used for static LHS actions you will generally suffix the name with *_STATIC* (i.e. *MY_APPLICATION_STATIC*).

Structure

```
<BUSINESS_PROCESS_DEFINITION>
  <SHORT_NAME>MY_BPD_STATIC</SHORT_NAME>
  <FULL_NAME>My Static BPD LHS</FULL_NAME>
  <DESCRIPTION>Business process to provide LHS workbasket action for my BPD.</DESCRIPTION>
  <STATIC_EVENT_LIST/>
  <OPERATION_LIST/>
  <ACTION_LIST/>
</BUSINESS_PROCESS_DEFINITION>
```

Static Events

There is a trigger defined on the business_process_definitions table that fires on insert or update of a business process definition. This trigger calls a procedure to run any static events defined in the business process. To define a static event you provide the name of the event as shown in the markup below:

Static Event BPD Markup

```
<STATIC_EVENT_LIST>
  <STATIC_EVENT>
    <EVENT_LABEL>MY_STATIC_EVENT_LHS</EVENT_LABEL>
  </STATIC_EVENT>
</STATIC_EVENT_LIST>
```

You can define multiple static events in a single business process. Each event will map to an operation which defines the work that will be carried out when the event is run.

Note: There is no delete trigger which means that if you need to remove your static business process you should first update the business process definition and remove the actions and then delete the record.

Operations

For each static event you will define an associated operation. This operation will end and create one or more actions and define the workbaskets the actions will be made available to.

Operation BPD Markup

```
<OPERATION_LIST>
  <OPERATION name="MY_OPERATION_LHS">
    <OPERATION_NAME>MY_OPERATION_LHS</OPERATION_NAME>
    <EVENT_LABEL>MY_STATIC_EVENT_LHS</EVENT_LABEL>
    <DESCRIPTION>Left hand side link for xxxx.</DESCRIPTION>
    <NEW_WORKBASKET_LIST>
      <SYSTEM_PRIV wb_type="LHS">MY_PRIVILEGE</SYSTEM_PRIV>
    </NEW_WORKBASKET_LIST>
    <END_ACTION_LIST>
      <ACTION_MNEM use_current_wb_list="true">MY ACTION LHS</ACTION_MNEM>
    </END_ACTION_LIST>
    <NEW_ACTION_LIST>
      <ACTION_MNEM>MY ACTION LHS</ACTION_MNEM>
    </NEW_ACTION_LIST>
  </OPERATION>
</OPERATION_LIST>
```

The operation is associated with your static event using the *EVENT_LABEL*.

The *NEW_WORKBASKET_LIST* defines what workbaskets to make your action(s) available to. As with a delegation you can define workbaskets to be scoped to resource roles, role users, privileges, or queries. Generally when creating LHS links we scope them to a privilege. We then give this privilege to the roles which should have access to our LHS actions. To make the action visible on the LHS workbasket you must define the workbasket type to be *LHS*. You do this using the attribute *wb_type="LHS"* which is defined on the privilege.

On the operation being run we want to end all our workbasket actions (in case they have already been created) and recreate them. We define the actions to end and create in the *END_ACTION_LIST* and *NEW_ACTION_LIST*

respectively. These should contain the same set of actions to ensure actions are cleaned up correctly before recreating them. For each action in your end action list your should define the attribute *use_current_wb_list="true"*. This ensures that you only end actions associated with the workbaskets defined in your operation. If this attribute is missing all matching actions will be ended regardless of what workbasket they are associated with which is generally not the desired behaviour. The actions listed here reference actions defined in your business process.

Actions

Operations use actions defined in your business process.

Action BPD Markup

```
<ACTION_LIST>
  <ACTION>
    <ACTION_MNEM>MY ACTION LHS</ACTION_MNEM>
    <DEFAULT_PROMPT>Launch Application</DEFAULT_PROMPT>
    <COMMENT>Entry into my application</COMMENT>
    <ACTION_SOURCE_CODE>
      <call-module module="MY_APPLICATION" type="modal" theme="new"/>
    </ACTION_SOURCE_CODE>
  </ACTION>
</ACTION_LIST>
```

Each action will define an *ACTION_MNEM* which is used by the operation to reference the action. You define a prompt which is visible to the user and a comment to describe the action. Finally you define your *ACTION_SOURCE_CODE* which is FOX module code that will be run when the action is clicked. This will generally be used to call into a module allowing the user to access those applications for which they have be granted the relevant privileges.

Exercises

1. Create a static BPD that can be used to launch your planning application FOX module using the create entry theme.
2. In your operation scope your workbasket to a privilege named *XX_TRAINING_PLAN_APP_LHS* replacing *XX* with your initials.
3. Add your static business process to the business_processes and busines_process_definitions tabls.
4. Grant your privilege to the *PLAN_APP_APPLICANT* role in your team.

Training:Architecture:Business Process Training Worksheets:Starting Business Process

Introduction

Now that we have prepared enough of our business process definition we are ready to start the business process. This section will explain concepts involved in starting a business process.

Concepts

Before you start a business process routine you need at a minimum an entry stage, a standard stage, a transition between the two and an operation. On starting a business process run a business routine will be created, a business context will be created and associated with the routine, all process assignments and external assignments (explained later) will be created and scoped to the business routine context, the entry stage will be created and its sole transition will be run.

Starting the Routine

A routine of the business process is started by “raising an event”, which is done by calling one of the “new_event” procedures/functions defined in the bpm_update package. You provide this function with the name of the event that starts the business process routine and context XML which is used by the event to create the business context. The context XML should contain the business context you will be using with your business process. You can also include further elements for the event to use when creating your business context.

```
--create context and start workflow
SELECT
  XMLELEMENT("EVENT"
  , uctx.createcontext(l_id||'MY_UREF', 'PRIMARY_DATA')
  )
INTO l_context
FROM dual;

l_new_event_xml := bpmmgr.bpm_update.new_event_xml(
  p_event_label => 'MY_BPD_CREATE'
, p_wua => p_wua_id
, p_context => l_context
, p_calling_module_code => 'my_package.my_procedure'
, p_calling_comment => 'Create new business routine for MY_BPD.'
);
```

On starting a business routine the entry stage's transition will be run. This may lead to further subsequent transitions being run. If any of the after stages in these transitions were marked up with *<RETURN_ENVIRONMENT>true</RETURN_ENVIRONMENT>* they will return the workbasket action XML associated with the stage on completion of the function call (i.e. into the l_new_event_xml variable shown in the function call above). The code below shows this markup.

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE1a</TRANSITION_LABEL>
```

```
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE1</STAGE_LABEL>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
      <RETURN_ENVIRONMENT>true</RETURN_ENVIRONMENT>
    </AFTER>
  </MOVE>
</TRANSITION>
```

In the above example if the transition *STAGE1a* is run by our event then the workbasket action XML associated with STAGE10 will be returned by the function.

Operation

The event raised when starting a business process routine is associated with an operation. The operation will define a query that creates the root business context for the routine using the context XML passed in when raising the event. A sample operation is shown below.

```
  <OPERATION>
    <OPERATION_NAME>MY_BPD_CREATE</OPERATION_NAME>
    <EVENT_LABEL>MY_BPD_CREATE</EVENT_LABEL>
    <DESCRIPTION>Entry point for My BPD workflow</DESCRIPTION>
    <NEW_PROCESS_LIST>
      <CALL_QUERY>
        <SQL>
XMLELEMENT("PROCESS"
, XMLELEMENT("PROCESS_SHORT_NAME", 'MY_BPD')
, UCTX.addPurpose(
    UCTX.extractContext(bpm.event_xml, 'PRIMARY_DATA')
      , 'PRIMARY_DATA'
      , 'WORKBASKET'
    )
, XMLELEMENT("CONTEXT_NAME", 'MY_BPD_ROOT')
)
        </SQL>
      </CALL_QUERY>
    </NEW_PROCESS_LIST>
  </OPERATION>
```

This operation will create a new routine with a business context named *MY_BPD_ROOT* with a single UREF with a purpose of PRIMARY_DATA.

Panel Actions

A common pattern is to start a case and then immediately enter a module where the case can be worked on. When entering a module from the workbasket you have the business stage ID available in the :{env} DOM to display any panel actions you may have. When entering a module from another module after starting your business process the business stage ID is not available. This means that your panel actions will not be displayed.

To get around this we can use the event XML passed back from the new_event_xml function that we used to call the event that created the business process routine. The event xml will contain the workbasket action XML of the after stages marked up with *<RETURN_ENVIRONMENT>true</RETURN_ENVIRONMENT>* of any transitions that were run on creating our business process routine. We can return this from our procedure to our calling module. Our calling module can then pass it in as a parameter when calling the module where the case will be worked on. You can then place the workbasket action XML into the environment DOM ensuring that the correct business stage ID is available when you call the action to setup your panel list.

The code below will create our case returning the new ID, context and event XML to the :{temp} DOM.

```
  <fm:api name="api-create-app">
    <fm:statement>
DECLARE


  l_wua_id NUMBER := :wua_id;


  l_new_id NUMBER;
  l_context XMLTYPE;
  l_event_xml XMLTYPE;


BEGIN


  my_schema.my_package.create_new_case(
    po_new_id => l_new_id
  , po_context => l_context
  , po_event_xml => l_event_xml
  );


  :new_id := l_new_id;
  :context := l_context;
  :event_xml := l_event_xml;


END;
    </fm:statement>
    <fm:using name=":wua_id">:{user}/WUA_ID</fm:using>
    <fm:using name=":new_id" direction="out" datadom-type="string" sql-type="varchar" datadom-location=":{temp}/ID"/>
    <fm:using name=":context" direction="out" datadom-type="dom" sql-type="xmltype" datadom-location=":{temp}"/>
    <fm:using name=":event_xml" direction="out" datadom-type="dom" sql-type="xmltype" datadom-location=":{temp}"/>
  </fm:api>
```

We then use these values when calling into our editing module.

```
<fm:call-module module="CASE_EDITING_MODULE" theme="edit" type="modal-replace-this-preserve-caller-callbacks-for-exit" params=":{temp}/*"/>
```

On entry into our module we then copy the workbasket action XML to the :{env} DOM ensuring that the business stage ID is in place for setting up our panel actions.

```
<fm:copy from=":{params}/WORKBASKET_ACTION/*" to=":{env}/WORKBASKET_ACTION"/>
```

One point to note is if you go through multiple stages before returning from the event (E.g. conditional stages or sync stages), if any of the transitions to these stages are marked up with *<RETURN_ENVIRONMENT>true</RETURN_ENVIRONMENT>* then each one of these will return their workbasket action XML. This may cause problems when you try to display your panel actions as it will not know which business stage ID to use.

Exercises

1. Add an operation to your business process definition that will create the root business context for your routine.
2. Modify transition *PLNAPP1a* so that the after stage will return its workbasket action XML.
3. Modify the trainingmgr.(initials)_plan_app.create_new_application function so that as well as creating your application it starts the business process routine and returns the workbasket action XML to the calling module along with any other parameters you need. (Note: if you are returning multiple values you will need to change the function to a procedure with multiple OUT parameters).
4. Modify your module so that it calls the create_new_application procedure and uses the values it returns as parameters when calling into your editing module.
5. Change your edit entry theme so that it ensures the correct business stage ID is available in the :{env} DOM.

When you enter the edit application module on creating the business process you should have the panel actions you created earlier displayed. If you don't then check that you have your business stage ID in the :{env} DOM and that you have correctly implemented the panel actions as shown in the Panel Actions training sheet.

Training:Architecture:Business Process Training Worksheets:External Events

Introduction

Most of the time requests to transition the business process will be requested internally. This section will explain how to manually transition the business process.

External Transitions

You can use the new_event function in your code when needing to call a transition from outside of the business process. You can do this in TOAD using the transition name and your business context. You can also use a transition alias which can be marked up on one or more transitions in your business process. An example of this markup is shown below.

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE1a</TRANSITION_LABEL>
  <TRANSITION_ALIAS_LIST>
    <TRANSITION_ALIAS>MY_EXTERNAL_TRANSITION</TRANSITION_ALIAS>
  </TRANSITION_ALIAS_LIST>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE1</STAGE_LABEL>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE2</STAGE_LABEL>
    </AFTER>
  </MOVE>
</TRANSITION>
```

The advantage of using a transition alias is that you do not need to know exactly what stage the business process is in. E.g. suppose we have a business process that could be either stage 1 or 2 and an external event requires us to move to stage 3. So long as we know the business process will always be in stage 1 or 2 at this point we can place the same transition alias on each transition from stage 1 to stage 3 and stage 2 to stage 3. We can then raise an event passing in the transition alias and the correct transition will run.

An example of how to call an event externally is shown below.

```
DECLARE

  l_dummy XMLSEQUENCETYPE;
  l_context XMLTYPE;

BEGIN

  SELECT
    xmlelement("EVENT"
      , uctx.createcontext(:p_uref, 'PRIMARY_DATA')
```

```
    )
  INTO l_context
  FROM dual;

  l_dummy := bpmmgr.bpm_update.new_event(
    p_event_label => :event_label
  , p_wua => 1
  , p_context => l_context
  , p_calling_module_code => 'pl/sql'
  );

END;
```

The event label can be any one of a transition label, transition alias, or operation (used when starting the business process). Together the event label and business context should identify a single transition from one stage to another. This means you cannot use it with a transition alias that maps to multiple active stages in the business process. More subtly this means that you cannot use it with a transition join as the business process does not know which of the before stages is the invoking stage.

Use in Debugging

You can use the new_event function to refresh the assignments that have been made on a stage. To do this you can create a temporary transition that uses the stage as its before and after stage. This will end and recreate the workbasket actions for that stage. If the transition does not exist you can add it to your business process definition temporarily and it will be available for you to use immediately, removing it later when it is no longer required.

Training:Architecture:Business Process Training Worksheets:Putting Into Practice

Introduction

We will now look to further implement the planning application business process. Ensure that as you go through the steps below you are changing the markup on your workflow diagram once your have implemented the stages or transitions and that your new actions are available and working as expected.

Exercises

PLNAPP10

1. Test the workbasket actions for your applicant assignment that we previously implemented. These should include entering your module, cancelling the business process, submitting the business process and reassigning the business process.
2. Test the viewer action to enter your module under the view entry theme.

PLNAPP20

1. Add an action for your applicant to view the submitted planning application in a background workbasket *Submitted Applications*.
2. Add an action for viewers to view the submitted planning application in a background workbasket *Submitted Applications*.

PLNAPP40

1. Add an action for case managers to view the submitted application before anyone has taken ownership of it.
2. Add an action for the case manager that has taken ownership to view the submitted application.
3. Add an action to send the case for review (for now send it to PLNAPP60 as PLNAPP50 is a subroutine which will be explained later in the training).

PLNAPP50

Allow a case manager to add a decision. To do this you need to:

1. Add a new *decision* entry theme to your application's FOX module.
2. Show a read only view of your application
3. Add an action that will change the status of your application to *APPROVED* and another action that will change the status of your application to *REJECTED*. You can use the trainingmgr.xx_plan_app.update_application_status procedure to do this.
4. Add a workbasket action *Add Decision* that will launch your module using the decision entry theme.

Training:Architecture:Business Process Training Worksheets:Conditional Stages

Introduction

This section will introduce you to how decision stages are used in the business process.

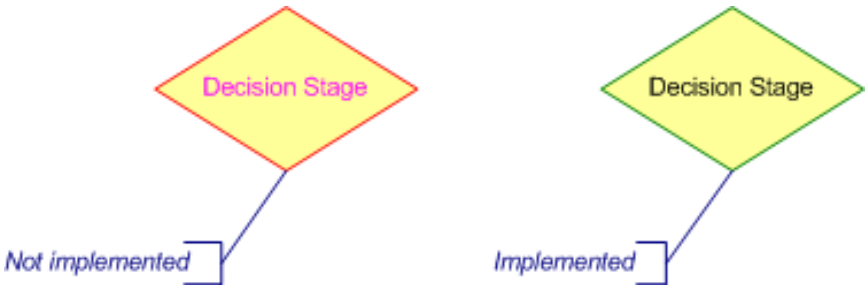
Concepts

Decision stages are used by the business process when we need to decide which path to take in the workflow between two or more different paths. Unlike a fork where we will go down every path a decision stage will only ever take one of the paths.

The path a decision stage takes is controlled by a condition. This condition returns true or false and can take any of *QUERY_NAME|WHEN|AND|OR|NOT|NO* elements as defined in the Run API training sheet.

For a decision stage you define a number of cases that could occur. Each of these cases include a transition and a condition list. If the conditions in the condition list evaluate to true the transition will be run. For each case you can optionally provide a message to be displayed to the user that invoked the event if the case's condition evaluates to true. You can also define a transition to be run if all the cases evaluate to false for which you can provide a message to be displayed if the transition is run.

Workflow Markup



BPD Markup

```
<CONDITION_STAGE_LIST>
  <STAGE>
    <STAGE_LABEL>STAGE1</STAGE_LABEL>
    <STAGE_TITLE>My First Decision Stage</STAGE_TITLE>
    <CASE_LIST>
      <CASE>
        <TRANSITION_LABEL>STAGE1a</TRANSITION_LABEL>
        <CONDITION_LIST>
          <QUERY_NAME>My Condition Query Definition</QUERY_NAME>
        </CONDITION_LIST>
        <ON_CASE>
          <MESSAGE_LIST>
            <MESSAGE>Case 1 transition run.</MESSAGE>
          </MESSAGE_LIST>
```

```
        </ON_CASE>
      </CASE>
      <CASE>
        <TRANSITION_LABEL>STAGE1b</TRANSITION_LABEL>
        <CONDITION_LIST>
          <WHEN>my_schema.my_package.count_stuff(my_id) = 1</WHEN>
        </CONDITION_LIST>
        <ON_CASE>
          <MESSAGE_LIST>
            <MESSAGE>Case 2 transition run.</MESSAGE>
          </MESSAGE_LIST>
        </ON_CASE>
      </CASE>
    </CASE_LIST>
    <OTHERWISE_TRANSITION_LABEL>STAGE1c</OTHERWISE_TRANSITION_LABEL>
    <ON_OTHERWISE>
      <MESSAGE_LIST>
        <MESSAGE>Otherwise transition run.</MESSAGE>
      </MESSAGE_LIST>
    </ON_OTHERWISE>
  </STAGE>
</CONDITION_STAGE_LIST>
```

Exercises

1. Add the decision stage *PLNAPP70*. The condition should check the status of the application. If *APPROVED* we go to stage *PLNAPP80*, if *REJECTED* we go to stage *PLNAPP90*, if neither of these we will go back to stage *PLNAPP60*. (There are several ways of implementing this. If you so choose you can use a function provided in your planning application package *get_application_status(p_application_id)* to retrieve the application status)
2. Provide an appropriate message to the user depending on the transition run.
3. Add a workbasket action to *PLNAPP60* to transition the workflow on to *PLNAPP70*.

Training:Architecture:Business Process Training Worksheets:Sync Stages

Introduction

The following section will introduce you to sync stages and what they are used for.

Concepts

Sync stages are generally used to tidy the workflow diagram, when dealing with transition forks and joins, or to provide a single set of tasks to a number of transitions that have the same after stage.

A sync stage has no actions and will have a single transition out of it. On entering a sync stage this transition is automatically called.

Tidying The Workflow Diagram

You may transition to a sync stage to make a workflow diagram clearer. This can help when you have two stages that are in very different parts of your diagram that need to be linked by a transition. An example of this is in the *SPIRE_BPD_DIAGRAM* workflow on the *SIEL(S)* tab. Here a sync stage *S49* is used when transitioning between stages *S3* and *S50*. As you can see from the diagram drawing a transition between these two stages could make the diagram harder to understand.

Dealing With Transition Forks and Joins

If your workflow has multiple stages that need to go to the same stage (or stages) you can have the before stages transition to a sync stage so long as you know that only one of the stages is active at any one time. The sync stage will then transition to the after stage (or stages) they need to go to. This may be used for many-to-many transitions where only one of the before stages is active, or for joining together stages after a decision stage where again we know that only one stage is active.

An example of a many-to-many transition using a sync stage is on the *PED_LIC_TRANSACTION* workflow where stages *PEDTXN20*, *PEDTXN26* and *PEDTXN28* transition to sync stage *PEDTXN29* which subsequently goes to several after stages. Note that only one of *PEDTXN20*, *PEDTXN26* and *PEDTXN28* will be active at any one time. The sync stage is not joining them together, rather it is simplifying the workflow.

An example of the stages after a decision stage being joined back together is on the *PED_LIC_TRANSACTION* workflow where after decision stage *PEDTXN105* the stages *PEDTXN110* and *PEDTXN115* both go to sync stage *PEDTXN116* to join the workflow back together.

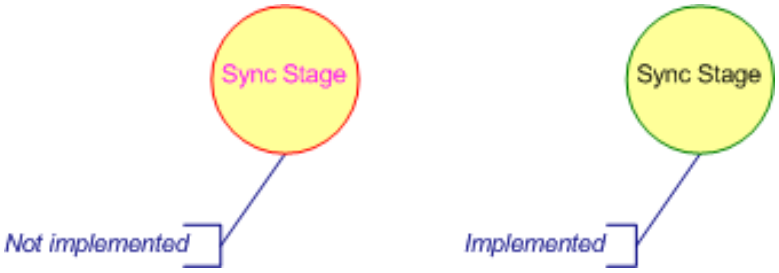
Both of these could be represented by transition joins but it may make the diagram clearer and the implementation easier using a sync stage if we know that only one of the stages will be active and hence that they do not actually need joining together.

If you need to select the parent context for one of the before stages this is not possible when going directly into a transition join. Here you may transition to a sync stage when selecting the parent context which subsequently runs the transition for the join. Parent and child contexts are explained later in the training.

Provide Common Tasks

If a number of stages transition to the same after stage and each transition needs to perform the same tasks you can centrally manage this by having each stage transition to a sync stage. The sync stage will then contain the tasks (i.e. APIs, emails, assignments) that need to be carried out. This should not be used in place of a transition join. A join will be used to join together divergent paths in the workflow. A sync stage will not be joining together active paths as it will only continue the workflow on for the stage that transitions into it, rather it is just used to centrally manage the tasks the transition will perform.

Workflow Markup



BPD Markup

```
<SYNC_STAGE_LIST>
  <STAGE>
    <STAGE_LABEL>STAGE1</STAGE_LABEL>
    <STAGE_TITLE>My first sync stage</STAGE_TITLE>
  </STAGE>
</SYNC_STAGE_LIST>
```

Exercises

1. Implement sync stage *PLNAPP90* and the subsequent transition join. The transition should run an API to update the status of the application to *CLOSED*.
2. Add a workbasket action for the document manager on *PLNAPP80* named create documents that will transition the workflow on to *PLNAPP90*. Notice that in this stage you do not have to take ownership but an assignment is automatically made for you to one of your two document managers. This is due to using an assignment that makes the assignment for you on *FIRST_USE*. As it is a *POOL* assignment, the assignment will be scoped to one of the *ROLE_USERS* in the *DOCUMENT_MANGER* role.

Training:Architecture:Business Process Training Worksheets:Process XML

Introduction

The following section will introduce you to the process XML package and how it is used.

Concepts

The process XML package is used to carry out application specific tasks or provide application specific information for generic components. This is done using UREFs by following a strict naming convention.

A process XML package is created in bpmmgr and uses a name in the format *PROCESS_XML_UREFTYPE* where *UREFTYPE* is substituted with the UREF type of the application that the package will provide application specific information for.

Each procedure or function in the process XML packages that performs the same function should have the same name. This means that all a generic component needs to work out in order to call the correct procedure or function for an application is the package to call. This can be done using dynamic SQL and the applications UREF.

Worked Example

Suppose we have two applications which have a function *get_string(my_id)* which returns a string that is needed by a generic component. The generic component needs to work with both applications and run the correct function depending on which application it is currently processing.

For application one we have defined a UREF type *APP1* and for application two we have defined a UREF type *APP2*. We can create two packages, *BPMMGR.PROCESS_XML_APP1* and *BPMMGR.PROCESS_XML_APP2* each of which will define the function *get_string(my_id)*.

The generic component can then use the application's UREF to call the correct function. The following code shows an example of how we can do this:

```
-- create pl/sql statement to call function
-- note that the called package is determined dynamically based off the UREF type
l_plsql := 'BEGIN :out_string := bpmmgr.Process_Xml_'||bpmmgr.uref.getUrefType(l_application_uref)||'.get_string(:in_id); END;';
-- execute the pl/sql statement
EXECUTE IMMEDIATE l_Plsql USING
  -- retrieve the returned string
  OUT l_out_string,
  -- pass in the application ID retrieved by removing the UREF type from the application UREF
  IN REPLACE(l_application_uref, bpmmgr.uref.getUrefType(l_application_uref));
```

In the above code the called package is worked out at runtime depending on the UREF that we are working with.

Using It In The Business Process

The business process makes use of this in several ways. Two common uses are to display application specific information on the central workbasket and to provide information about your application at the top of your application's FOX modules.

Workbasket

You can provide additional application specific information to be displayed in the central workbasket by providing a function in your application's process XML package.

```
  FUNCTION workbasket(
    p_key VARCHAR2
  , p_xml_params XMLTYPE DEFAULT NULL
  ) RETURN XMLTYPE
```

The above function is defined in your process XML package and returns an XMLTYPE with its root element set to *WORKBASKET*. The following is a sample of the element that can be included in the XML returned that the workbasket will set out.

```
<WORKBASKET>
  <!--Displayed in Transaction / Ref column on the workbasket-->
  <FOLDER_NUMBER/>
  <REF_NUMBER/>
  <WB_ICON/>
  <WB_PROMPT/>
  <WB_HINT/>
  <!--Displayed in Subject / Topic column on the workbasket-->
  <SUBJECT/>
  <TOPIC/>
  <ICTOPIC>
    <I/> <!--image url-->
    <P/> <!--image prompt-->
    <H/> <!--image hint-->
  </ICTOPIC>
  <!--Displayed in Company column on the workbasket-->
  <COMPANY/>
  <ICCOMPANY>
    <I/> <!--image url-->
    <P/> <!--image prompt-->
    <H/> <!--image hint-->
  </ICCOMPANY>
  <!--Displayed in Status / Date column on the workbasket-->
  <STATUS/>
  <ICSTATUS>
    <I/> <!--image url-->
    <P/> <!--image prompt-->
    <H/> <!--image hint-->
  </ICSTATUS>
  <DATE/>
```

```
</WORKBASKET>
```

Context Header

You can include a header on your FOX modules to show information about your application.

```
  FUNCTION context_header(
    p_key VARCHAR2
  , p_xml_params XMLTYPE DEFAULT NULL
  ) RETURN XMLTYPE
```

The above function will return an XMLTYPE with contextual information about your application. The root element is *CONTEXT* and it can contain up to six *SUBJECT* elements under it.

```
<CONTEXT>
  <SUBJECT>
    <VALUE/> <!-- defines the text displayed -->
    <DESC/> <!-- defines the text prompt -->
    <USE/> <!-- defines where it will be displayed in the grid-->
  </SUBJECT>
</CONTEXT>
```

The context header is displayed in a 2 x 3 grid. The *USE* element defines where in the grid the data is displayed taking the values shown in the grid below.

A1	B1	C1
A2	B2	C2

To query the context information into your module you need to have libraried in *LAYOUT1* and call an API to retrieve the context header XML. You can make use of the bpmmgr.uref.getprocessedxml(p_uref, p_result_type) function to do this as shown in the API call below:

```
<fm:api name="api-getContextData">
  <fm:statement>
BEGIN

  :result := appenv.uref.getprocessedxml(
    p_uref => :uref
  , p_result_type => 'context_header'
  );

END;
  </fm:statement>
  <fm:using name=":result" datadom-type="dom" sql-type="xmltype" direction="out" datadom-location=":{theme}/CONTEXT"/>
  <fm:using name=":uref">:{params}/CONTEXT/SUBJECT[USE/PURPOSE='PRIMARY_DATA']/REF_ID1</fm:using>
</fm:api>
```

Exercises

Workbasket Data

1. Create a process_xml package for your UREF type.
2. Add the workbasket(p_key, p_xml_params) function to it, returning an XML structure with relevant data from you application.

Context Header

1. Add the context_header(p_key, p_xml_params) function to your process_xml package, returning an XML structure with relevant data from you application.
2. Add an API in your module to retrieve the context header.
3. Call your API on entry into your module.

Training:Architecture:Business Process Training Worksheets:Signals

Introduction

This section will introduce you to signals and how they are used in the business process.

Concepts

A signal can be used to raise an event for all transitions that are triggered by that signal. Unlike a transition alias a signal can be used to transition multiple stages, the stages can exist either in the same business process or another business process.

A signal is raised by making a call to the bpm_update.signal function passing in the name of the signal, the ID of the root business context of the routine that is raising the signal, and the ID of the root routine that raised the signal. The root business context and routine can be obtained through the BPM package.

Signals are generally raised when a transition in a business process requires other transitions to be fired. In this case the signal is defined as a query definition and it is called as one of the tasks done by the transition. They are especially useful when you do not know exactly which transitions need to be called and there may be zero, one or more transitions so a transition alias, which must match exactly one transition, is not suitable.

To receive a signal a transition is marked up as being invoked by that signal. On a signal being raised the BPM will look for any active stages that have transitions that are invoked by the signal that was raised. These transitions may exist in the same business process routine, the same business process in a different routine, or an entirely different different business.

BPD Markup

Raising a signal

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE1a</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE1</STAGE_LABEL>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE2</STAGE_LABEL>
    </AFTER>
  </MOVE>
  <IN_TRANSITION>
    <API_LIST>
      <QUERY_NAME>Signal Something Happened</QUERY_NAME>
    </API_LIST>
  </IN_TRANSITION>
</TRANSITION>
```

```
<QUERY_DEFINITION>
  <QUERY_NAME>Signal Something Happened</QUERY_NAME>
  <SQL>
DECLARE
  l_result XMLSequenceType;
BEGIN
  l_result := bpm_update.signal(
    p_signal => 'SIGNAL_SOMETHING_HAPPENED'
  , p_seed_bc_id => bpm_update.get_parent_bc_id(bpm.ROOT_CONTEXT)
  , p_seed_br_id => bpm.ROOT_ROUTINE
  );
END;
  </SQL>
</QUERY_DEFINITION>
```

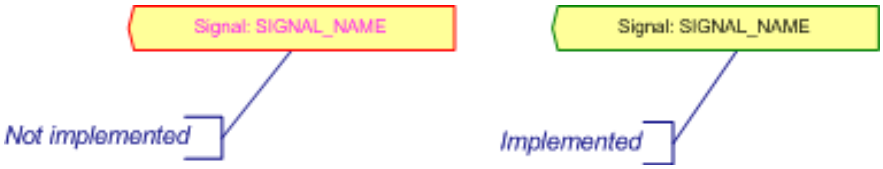
Receiving a signal

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE50a</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE50</STAGE_LABEL>
      <INVOKER_SIGNAL>MUST-EXIST</INVOKER_SIGNAL>
      <AUTO_INVOKE_FOR>
        <SIGNAL>SIGNAL_SOMETHING_HAPPENED</SIGNAL>
      </AUTO_INVOKE_FOR>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE60</STAGE_LABEL>
```

```
    </AFTER>
  </MOVE>
</TRANSITION>
```

Workflow Markup

Raising a signal



Receiving a signal

Exercises

1. Implement workbasket actions on stages *PLNAPP40* and *PLNAPP60* to return the application to the applicant. They should have the following properties:
   1. Provide a confirm dialog to the user.
   2. Implement the *PLNAPP40b* and *PLNAPP60b* transitions respectively.
   3. Raise the *AMEND_APPLICATION* signal in transition.
2. Implement transition *PLNAPP20a* to be invoked on receiving the *AMEND_APPLICATION* signal.

Training:Architecture:Business Process Training Worksheets:Emails

Introduction

The following section will show how you can send emails from the business process.

Concepts

As well as running APIs and making assignments another common task that you may want to perform during a transition is to send an email to one or more assignments or a custom group of users. The business process contains markup that allows you to send basic emails to targeted recipients.

BPD Markup

```
<EMAIL_LIST>
  <EMAIL>
    <EMAIL_ASSIGNMENT_LIST>
      <WORKBASKET_ASSIGNMENT>
        <ASSIGNMENT>MY_ASSIGNMENT</ASSIGNMENT>
        <ASSIGNMENT_GROUP>ASSIGNEE-WHEN-UNASSIGNED</ASSIGNMENT_GROUP>
      </WORKBASKET_ASSIGNMENT>
    </EMAIL_ASSIGNMENT_LIST>
    <EMAIL_USER_LIST>
      <QUERY_NAME>Get Email Users</QUERY_NAME>
    </EMAIL_USER_LIST>
    <SHOW_COPY_RECIPIENTS>true</SHOW_COPY_RECIPIENTS>
    <SUBJECT>
      <SUFFIX_TEXT>New Work Item</SUFFIX_TEXT>
    </SUBJECT>
    <BODY>
      <SIMPLE_CONTENT>Email body goes here.</SIMPLE_CONTENT>
    </BODY>
  </EMAIL>
</EMAIL_LIST>
```

Email recipients

As shown in the above markup we can specify the recipients for an email either by using workbasket assignments or through the *EMAIL_USER_LIST*. Workbasket assignments work in much the same way as for action sets, except you do not define a workbasket type. The *EMAIL_USER_LIST* is defined using the same markup as the delegation assignment markup. It can accept *CURRENT_USER|ROLE_USERS|RESOURCE_ROLE|SYSTEM_PRIV|QUERY_NAME|SQL*. These are explained on the Delegations and Process Assignments worksheet.

Email Subject

The email subject is defined in your process_xml package. You can define a special function with the signature *email_subject_prefix(p_key IN VARCHAR2, p_xml_params IN XMLTYPE DEFAULT NULL) RETURN VARCHAR2* to return a string to use as the email subject. If this function is not defined then the subject defaults to using the *FOLDER_NUMBER*, *REF_NUMBER* and *TOPIC* from your workbasket query. If the workbasket query is also not defined then the business process will raise an application error. You can provide additional text to be placed as a suffix to the subject returned by process_xml.

Email body

The email body can take either plain text using the *SIMPLE_CONTENT* element, or you can provide an HTML email body by using the *HTML_CONTENT* element instead. This will allow you to use HTML tags within your email body.

DEV Emails

You can view emails that would be sent via the *View Portal Emails* link in the *Admin Console* (use login *cfbrown* to access the admin console). Alternatively, you can view the email subject and body in the text_attachment column of the portalmgr.transmission_content table. If your email has just gone out you can generally find it easily by sorting the mail_id column in descending order and looking at the most recent emails sent.

Exercises

Add Emails to BPD

1. Send an email informing the applicant the application has been returned on transition *PLNAPP20a*.
2. Send an email informing the applicant their application was rejected on *PLNAPP70b*.
3. Send an email informing the applicant their application was approved on *PLNAPP80a*.

Provide Custom Subject

1. Add a procedure to your process_xml package to provide a custom subject.
2. Look either in the admin console or portalmgr at the email and compare the email subject. You should see in the earlier emails that the subject was constructed from the data returned by the process_xml_xxx.workbasket() function and the later emails used your custom string. For both emails part of the subject prefix is fixed whatever method you used.

Training:Architecture:Business Process Training Worksheets:Subroutines

Introduction

The following section will introduce subroutines and how they are used in the business process.

Concepts

You may find at times that a subset of a business processes activities are common to more than one business process. Rather than implementing the same activities in many business processes it may be possible in these instances to extract the similar activities out to make their own business process which can be reused across many different business processes.

This is done through the use of subroutines. A subroutine is defined in much the same way as a standard business process, except that it allows you to pass in assignments and tallies (tallies are explained later in the training) from its superroutine (the routine that invoked it). It can also provide feedback to its superroutine on completion of the routine.

Scenario

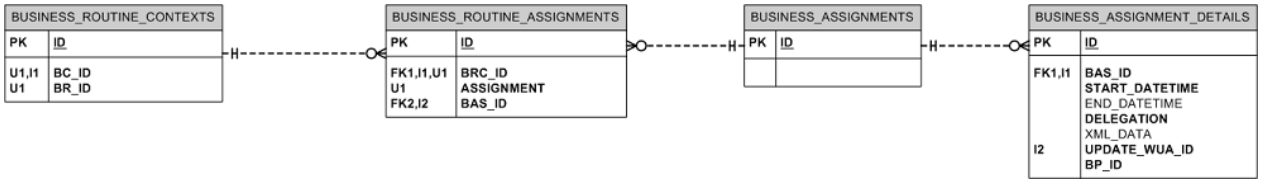
We have identified a need for our planning application to undergo a review before a decision is made. We currently have a very similar form of review taking place in a business process for an HR application. As this is a common requirement that may be used again in future business processes we have decided that instead of duplicating the code to carry out the review in our planning application we will create a new generic review business process. It should be possible for the review business process that we create to be implemented by our planning application business process, replace the review in the business process for our HR application, and be reused in future applications that are developed as required.

External Assignments

As a subroutine is an extension of another business process you may want to use the same assignments from your superroutine in your subroutine. For example, in our generic review process suppose we have a review manager whose responsibility it is to allocate the reviewers. We want the review manager to be the same as our case manager from our planning application business process. We could create a new process assignment in our subroutine for our case manager but that would restrict its ability to be reused as the review manager assignment would then be scoped to our planning application team's case manager role. Additionally we would then also need some way to keep the assignments in sync, so that if the case manager reassigned the case to another case manager the review manager assignment would also be updated.

External assignments allow us to pass in a reference to an assignment in our superroutine to our subroutine. This allows us to use different assignments in our subroutine depending on what business process it has been invoked from. Additionally, as both our subroutine and our superroutine reference the same assignment if the assignment scope changes it will change for both routines.

BPM Assignment Data Model



On creating an external assignment for a business routine a new business_routine_assignment is created and associated with our business_routine_context. The business_assignment it is associated with is the same assignment record from our superroutine. This means we use the same delegation and will have the same assignment scope as our superroutine. All we need to define our external assignment then is the name of our assignment as shown below:

External Assignment BPD Markup

```
<EXTERNAL_ASSIGNMENT_LIST>
  <EXTERNAL_ASSIGNMENT>
    <ASSIGNMENT>MY_EXTERNAL_ASSIGNMENT</ASSIGNMENT>
    <ASSIGNMENT_TITLE>My first external assignment</ASSIGNMENT_TITLE>
  </EXTERNAL_ASSIGNMENT>
</EXTERNAL_ASSIGNMENT_LIST>
```

Invoking a Subroutine

We can define markup against a stage in our business process definition to invoke one or more subroutines. The subroutine will be created on entering the business stage. We also specify any assignments defined in our superroutine that we want to pass into our subroutine.

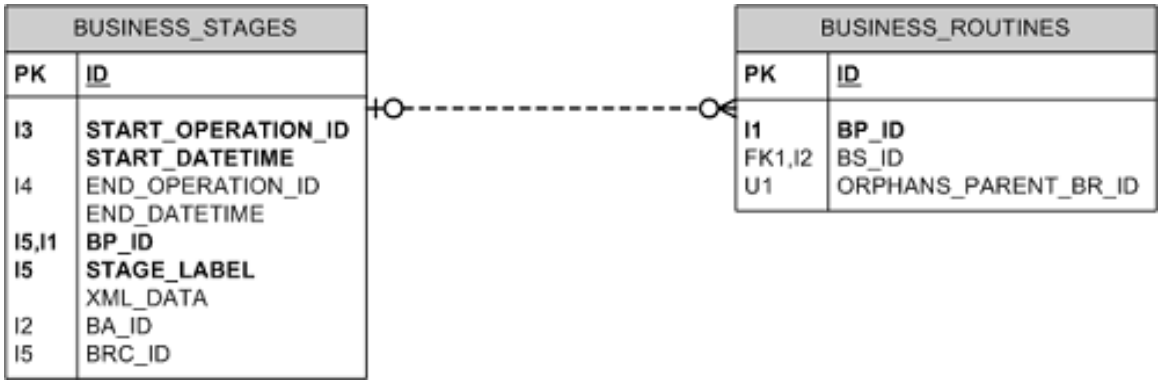
Subroutine BPD Markup

```
<STAGE>
  <STAGE_LABEL>MY_SUPERROUTINE_STAGE</STAGE_LABEL>
  <STAGE_TITLE>My Superroutine Stage</STAGE_TITLE>
  <ACTION_SET_LIST>
     ...
  </ACTION_SET_LIST>
  <SUBROUTINE_LIST>
    <SUBROUTINE>
      <SUBROUTINE_SHORT_NAME>MY_SUBROUTINE_BPD_NAME</SUBROUTINE_SHORT_NAME>
      <SUBROUTINE_ASSIGNMENT_LIST>
        <SUBROUTINE_ASSIGNMENT>
          <ASSIGNMENT>SUPERROUTINE_ASSIGNMENT_NAME</ASSIGNMENT>
          <INTERNAL_ASSIGNMENT>SUBROUTINE_EXTERNAL_ASSIGNMENT_NAME</INTERNAL_ASSIGNMENT>
        </SUBROUTINE_ASSIGNMENT>
      </SUBROUTINE_ASSIGNMENT_LIST>
      <SUBROUTINE_BLOCKED_TRANSITION>
        <PRIORITY>1</PRIORITY>
        <MESSAGE>You cannot do this because .....</MESSAGE>
        <ACTION_SOURCE_CODE>
          <call action="action-do-something"/>
        </ACTION_SOURCE_CODE>
      </SUBROUTINE_BLOCKED_TRANSITION>
    </SUBROUTINE>
  </SUBROUTINE_LIST>
```

```
</STAGE>
```

A subroutine will hold a reference to the business stage that created it as shown in the data model diagram below.

BPM Stage Subroutine Data Model



Before the business stage can be ended (i.e. by a transition to another stage) all of its subroutines must be complete. For a subroutine to be complete this generally means that all of its stages must be ended. You can also define stages in your subroutine as having a status of *INACTIVE* (see the Standard Stages worksheet for details on stage status). If an incomplete subroutine has only *INACTIVE* stages then when attempting to end its superroutine stage an *END_SIGNAL* will be sent to the subroutine's stages. As long as all *INACTIVE* stages define a transition that is invoked by *END_SIGNAL* that ends the *INACTIVE* stage by going to an exit stage or orphan stage (explained later in the training) the subroutine will complete and the superroutine stage can be ended.

If a subroutine is not complete or it is not possible to subsequently complete it when trying to end the superroutine business stage then the transition in the superroutine will fail. You can define a message to display to the user if this occurs and optionally supply some action code to run. This is done using the *SUBROUTINE_BLOCKED_TRANSITION* element shown in the XML markup above. The priority is used to determine which message and action code to run if multiple incomplete subroutines are found.

Subroutine Feedback

On entering an exit stage in a subroutine you can have it update the status of the stage in its superroutine that invoked it. You can use this new status to automatically invoke a transition in the superroutine. This is commonly used to move the superroutine on when the subroutine is complete. You can use different statuses to provide different feedback as required to invoke different transitions. For example you may update the status to *ABORTED* if the subroutine was cancelled and *COMPLETE* if the subroutine was completed. Each of these statuses may result in a different transition being fired by the superroutine.

Subroutine Feedback BPD Markup

```
<EXIT_STAGE_LIST>
  <STAGE>
    <STAGE_LABEL>STAGE998</STAGE_LABEL>
    <STAGE_TITLE>End (Aborted)</STAGE_TITLE>
    <FINAL_SUPER_ROUTINE_STATUS>ABORTED</FINAL_SUPER_ROUTINE_STATUS>
  </STAGE>
  <STAGE>
    <STAGE_LABEL>STAGE999</STAGE_LABEL>
    <STAGE_TITLE>End (Complete)</STAGE_TITLE>
    <FINAL_SUPER_ROUTINE_STATUS>COMPLETE</FINAL_SUPER_ROUTINE_STATUS>
  </STAGE>
```

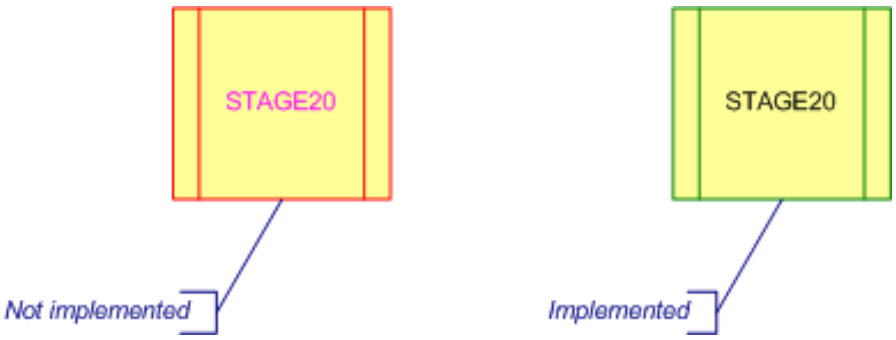
```
</EXIT_STAGE_LIST>
```

Invoking Superroutine Transition BPD Markup

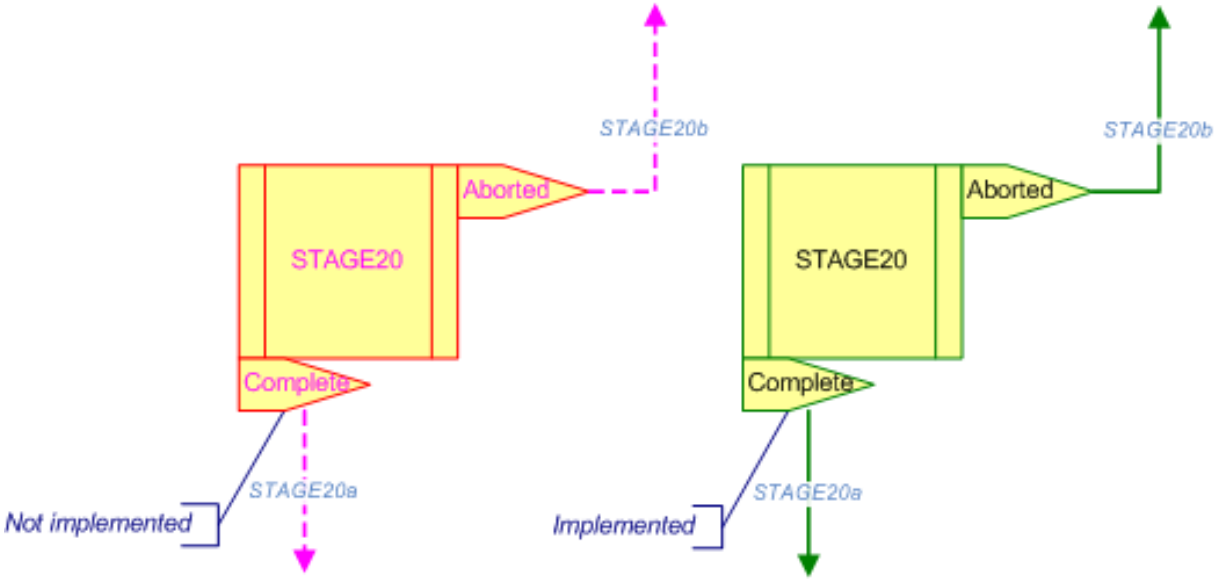
```
<TRANSITION>
  <TRANSITION_LABEL>STAGE20a</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE20</STAGE_LABEL>
      <AUTO_INVOKE_FOR>
        <STAGE_STATUS>COMPLETE</STAGE_STATUS>
      </AUTO_INVOKE_FOR>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE30</STAGE_LABEL>
    </AFTER>
  </MOVE>
</TRANSITION>
<TRANSITION>
  <TRANSITION_LABEL>STAGE20b</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE20</STAGE_LABEL>
      <AUTO_INVOKE_FOR>
        <STAGE_STATUS>ABORTED</STAGE_STATUS>
      </AUTO_INVOKE_FOR>
    </BEFORE>
    <AFTER>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
    </AFTER>
  </MOVE>
</TRANSITION>
```

Workflow Markup

Subroutine Without Feedback



Subroutine With Feedback



Exercises

Create Subroutine

Create a new BPD named *XX_REVIEW_SUBROUTINE* (replacing *XX* with you initials) with the following properties:

1. Has an external assignment called *REVIEW_COORDINATOR*.
2. Has entry stage *PLNAPPRVW1*, single standard stage *PLNAPPRVW2*, and single exit stage *PLNAPPRVW999*.
3. The entry stage should transition to the standard stage.
4. The standard stage should have a single action with an assigned workbasket assignment for the *REVIEW_COORDINATOR* assignment and prompt *Assign Reviewer*. The action should transition to the exit stage.
5. The exit stage should update the superroutine status to *COMPLETE*.

Invoke Subroutine

Modify your Planning Application BPD to invoke your subroutine by:

1. Markup stage *PLNAPP50* to invoke your new subroutine BPD.
2. Pass your case manager assignment into your subroutines external assignment.
3. Modify transition *PLNAPP40a* so that it moves to stage *PLNAPP50*.
4. Add an action to *PLNAPP60* to run transition *PLNAPP60c* which will pass back from decision stage to the review stage.
5. Add transition *PLNAPP50a* that is invoked when the stage status is set to *COMPLETE*.

Training:Architecture:Business Process Training Worksheets:Context Switches

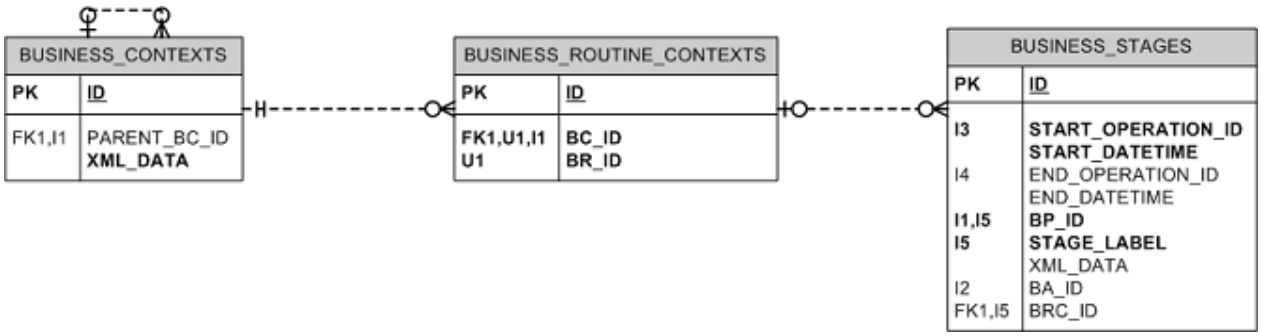
Introduction

The following section will show you how to create child contexts for a business process routine and discuss why they are used.

Concepts

A business process stage is linked to your business routine through the business routine context.

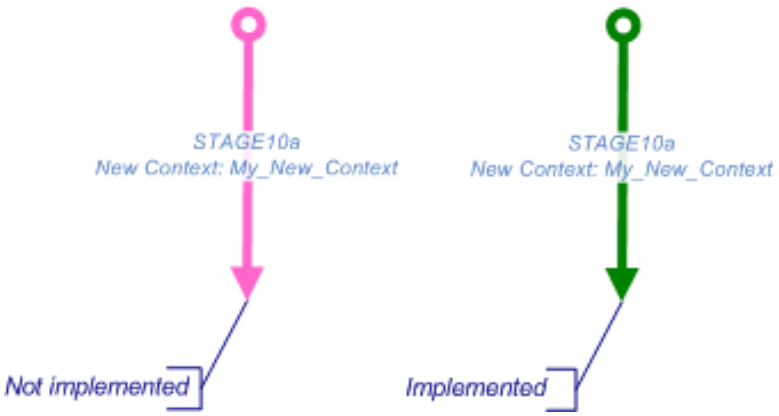
BPM Context/Stage Data Model



When running a transition the business routine context is used in conjunction with the stage label to identify the before stages that need to be ended. When creating the after stages in a transition they are by default linked to the business routine context that raised the transition. You can override this behaviour by specifying a new context for the after stage or stages to be linked to. This is required when you want to have more than one instance of the same stage active for a given business routine in order to be able to continue to identify it uniquely. If you try to transition to an after stage that already exists for your context then another stage will not be created, the existing stage will be used. You will see a common scenario where this is applicable later in the training when looking at Parallel Processing Patterns.

New Contexts

Workflow Markup



BPD Markup for New Context

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE10a</TRANSITION_LABEL>
```

```
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
    </BEFORE>
    <CONTEXT_SET>
      <NEW_CONTEXT_LIST>
        <CONTEXT_NAME>MY_NEW_CONTEXT</CONTEXT_NAME>
      </NEW_CONTEXT_LIST>
      <IN_CONTEXT>
        <API_LIST>
          <QUERY_NAME>Update New Context</QUERY_NAME>
        </API_LIST>
      </IN_CONTEXT>
      <AFTER>
        <STAGE_LABEL>STAGE20</STAGE_LABEL>
      </AFTER>
    </CONTEXT_SET>
  </MOVE>
</TRANSITION>
```

When moving to a new context you must specify the name of the context and the after stage or stages that will be created and linked to it. The name of the context can be either a static name (as shown) or dynamically generated by defining a *QUERY_NAME* (referencing a query in your business process) or *SQL* element. You can also optionally specify tasks to perform when the new context is created. These are similar to the tasks you perform *BEFORE\IN\AFTER* your transition such as making assignments, running APIs and emailing users.

Updating Your New Context

By default the new context will have the same XML as its parent context other than for the context name element. When the business process raises events, for example by clicking a workbasket action, this isn't a problem as the business process is aware of which business routine context initiated the event. When raising events externally however this presents a problem as we first need to try and identify the business routine context the event is running for based off the event itself and the context information passed into the event. If the event matches more than one of the business routine contexts found using the context xml provided you will receive an application error.

To get around this we can update our new context once it is created to make it unique. In the previous example we are calling a query that will update the context XML for our new context. This query is shown below.

BPD Markup for Updating a Business Context

```
    <QUERY_DEFINITION>
      <QUERY_NAME>Update New Context</QUERY_NAME>
      <SQL>
DECLARE
  l_new_uref VARCHAR2(10) := xp.get(bpm.event_xml, '/*/MY_NEW_UREF')
BEGIN

  /* make context unique by adding new uref as secondary data */
  bpm.set_context_xml(xp.set_xml(bpm.context_xml, '/*/CONTEXT',
uctx.addContext(l_new_uref, 'SECONDARY_DATA',
```

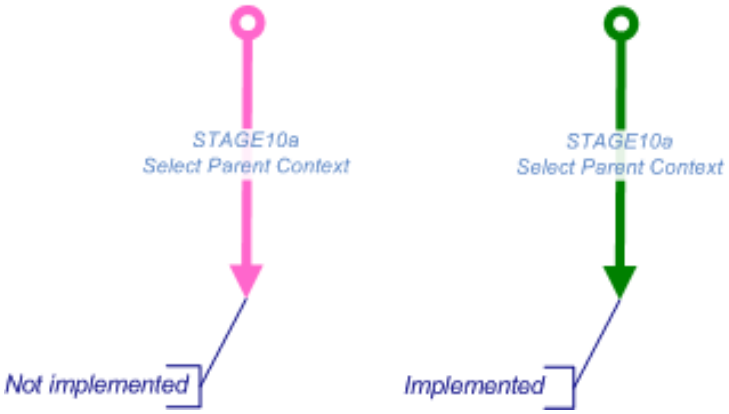
```
    uctx.removeContext('SECONDARY_DATA', bpm.context_xml))));

END;
      </SQL>
    </QUERY_DEFINITION>
```

In the above query we have passed a new UREF in through the event_xml which we then use to make the context unique by setting this UREF as the secondary data in our new context.

Parent Contexts

A new context will contain a reference back to its parent context. On finishing with a context you can move back to its parent context by running a transition and selecting the parent context when creating the after stages.

Workflow Markup



BPD Markup

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE10a</TRANSITION_LABEL>
  <MOVE>
    <BEFORE>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
    </BEFORE>
    <CONTEXT_SET>
      <SELECT_CONTEXT_LIST>
        <PARENT_CONTEXT/>
      </SELECT_CONTEXT_LIST>
      <AFTER>
        <STAGE_LABEL>STAGE20</STAGE_LABEL>
      </AFTER>
    </CONTEXT_SET>
  </MOVE>
</TRANSITION>
```

You will see an example of how this is used in the Parallel Processing Patterns worksheet.

Exercises

1. Add stage *PLNAPPRVW3* to your business process.
2. Change the transition from *PLNAPPRVW2* to move to *PLNAPPRVW3* in a new context.
3. In the new context run an API that updates the secondary data of the context XML to a person UREF (PER) of a person ID returned by the get_unassigned_reviewer(p_application_uref VARCHAR2) function in your package. Note: to get your application UREF you can use the uctx function uctx.getRefID1 passing in your business context which can be accessed using bpm.context_xml.
4. Add a single action with a prompt *Provide Response* which will to transition from *PLNAPPRVW3* to *PLNAPPRVW999* selecting the parent context. For now assign the action to your *REVIEW_COORDINATOR* assignment.

Training:Architecture:Business Process Training Worksheets:Contextual Assignments

Introduction

The following section will introduce you to contextual assignments.

Concepts

Contextual assignments are used when an assignment should be scoped to the business routine context rather than the entire business routine. Unlike process assignments and external assignments which are created at the point the business routine is created, contextual assignments are created on demand. Once created a contextual assignment will exist for the context it was created in and any of its child contexts.

Creating a Contextual Assignment

To create a contextual assignment you call the assignment.create_assignment function passing it the name of the contextual assignment and the name of the delegation profile that it will use. You can also optionally pass in your own assignment XML containing data that is in scope of the assignment and/or a varchar2 list of UREFs for the assignment to be scoped to in the case of assignments that are automatically made on *FIRST_CREATE* or *FIRST_USE*. You generally define the API creating the assignment in your business process in a query definition. You can then call this query as one of the tasks performed when running a transition or creating a new context.

BPD Markup for Creating an Assignment

```
    <QUERY_DEFINITION>
      <QUERY_NAME>Create Assignment</QUERY_NAME>
      <SQL>
DECLARE

  l_bas_id NUMBER;
  l_assignment_xml XMLTYPE;

BEGIN

  /* add data to assignment xml */
```

```
  SELECT
    XMLELEMENT("ASSIGNMENT"
    , XMLELEMENT("WUA_ID", bpm.wua_id)
    )
  INTO l_assignment_xml
  FROM dual;

  l_bas_id := assignment.create_assignment(
    p_assignment => 'MY_CONTEXTUAL_ASSIGNMENT'
  , p_delegation => 'MY_CONTEXTUAL_ASSIGNMENT_DL'
  , p_assignment_xml => l_assignment_xml
  );

END;
      </SQL>
    </QUERY_DEFINITION>
```

Contextual Assignment Delegations

As you pass the delegation profile to use into the create_assignment function you can define this parameter at run-time and use different delegation profiles each time the assignment is created. A more common pattern though is to use the same delegation profile which retrieves the valid UREFs for an assignement using a query. This query can retrieve the valid assignments using data you place inside your assignment XML, business context, or event XML if raising the event externally.

BPD Markup for Delegation Query Definition

```
    <QUERY_DEFINITION>
      <QUERY_NAME>Get Assignment Roles</QUERY_NAME>
      <SQL>
SELECT
  XMLTYPE(xmlelement("WORKBASKET"
  , xmlelement("SCOPE_UREF", rr.id)
  ).getclobval())
FROM decmgr.resource_roles rr
WHERE rr.id IN (
  SELECT rmc.rr_id
  FROM decmgr.resource_members_current rmc
  WHERE rmc.wua_id = xp.get_number(bpm.assignment_xml, '/*/WUA_ID')
)
      </SQL>
    </QUERY_DEFINITION>
```

In the above example we have defined a query definition that can be used by our assignment delegation to return the resource roles associated with the WUA ID that we placed in our assignment XML.

Other Functions

The assignment package contains some other functions that can be used to return information about an assignment or to manipulate the assignment. Two of the more useful functions are *remove_assignment* and *update_assignment*.

Removing Assignments

You can use the assignment.remove_assignment function to remove the assignment matching the assignment name that you pass in as a parameter. This can be done when the assignment is no longer required or before creating the assignment to ensure it is removed if it already exists.

Conditionally Scoping Assignment

Often you might like to use a delegation for more than one scenario with a contextual assignment. For example at times you may not want to have the assignment scoped and allow users to take ownership, hence you will need an *ASSIGNEE_WHEN_UNASSIGNED_LIST* that *NEVER* makes an assignment automatically. At other times you may know who you want to make the assignment to and want to make it right away. As the delegation profile is set to never automatically make assignments this will not happen when the assignment is created. Instead you can use the assignment.update_assignment function passing in your assignment and the assignment mode, which to make an assignment should be set to *ASSIGN*.

Exercises

1. Create a new delegation profile with the following properties
    1. Requires an assignment to be made at all times
    2. Automatically makes assignment on first use
    3. Does not use unassigned actions
    4. Uses a query definition to control valid assignments (for now just provide a name we will define the query later)
2. Create a contextual assignment named *REVIEWER* that uses your new delegation profile
3. Modify the API that updates your business context so that it also adds the person ID to the assignment XML and calls assignment.create_assignment passing in your new contextual assignment, associated delegation profile and the assignment XML you created.
4. Create the query definition that your delegation profile uses. This should return the WUA UREF (WUA) of the person matching the person ID in your assignment XML in the correct format (the required format is shown in the Assignments training worksheet).
5. Change your *Provide Response* action on *PLNAPPRVW3* to use your *REVIEWER* assignment instead of the *REVIEW_COORDINATOR* assignment.

Training:Architecture:Business Process Training Worksheets:Delay Stages

Introduction

This section will introduce delay stages and what they are used for.

Concepts

At times in your workflow you may need to wait for an event to occur either in another part of your business process or externally before proceeding. To do this we can move to a delay stage. This is a standard business process stage except that it has no actions defined. The business process will remain in this stage until moved on as a result of receiving a signal, an external event, or being one of the before stages in a transition invoked by another stage.

Workflow Markup

BPD Markup

```
<STAGE>
  <STAGE_LABEL>STAGE10</STAGE_LABEL>
  <STAGE_TITLE>Waiting for xxxxx</STAGE_TITLE>
</STAGE>
```

Training:Architecture:Business Process Training Worksheets:Parallel Processing Pattern

Introduction

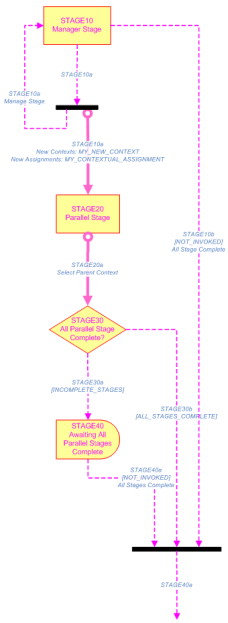
This training sheet will show an example of how we can put together some of the different business process elements we have learnt about so far to produce a pattern that can be used for parallel processing. The following example is not the only way this can be done and in some business processes you may need to use a different variant of it, but it should give you an idea of how you can use the different elements of a business process to provide this functionality.

Concepts

At times we may want to present the same set of stages to different users to work on independently of each other in a single business routine. It should be possible for them to end and create stages without affecting other users working on those stages. This is an example of parallel processing.

Used together child contexts and contextual assignments can be used to implement a parallel processing pattern where we have several instances of the same business process stage being worked on by different users at the same time. The example below shows you an example of how this can be implemented using a manager stage, decision stage, delay stage, and a transition join.

Workflow Example for Parallel Processing



Manager Stage

The *Manager Stage* is responsible for controlling the parallel workflow. It will have an action available to allow the manager to start a new parallel flow. This is shown as transition *STAGE10a* on the example diagram. This transition will create a new stage *STAGE20* within a new context. It will also recreate the manager stage so that the manager can retain access to add further parallel flows.

In our transition our new context should run an API to setup the new context and create our contextual assignment. The Context Switches and Contextual Assignments training sheets showed you how to do this.

Parallel Stage

There could be many instances of the parallel stage active for the same business routine at any one time. Each parallel stage will be linked to the business routine through a different business routine context. The business routine context along with the stage label allows us to uniquely identify each stage. In our example we show only one parallel stage but you can have any number of stages existing within the new context we created running in parallel with other flows using different instances of the same stages. When stages are created and ended within our new context this will have no impact on stages in other contexts.

Transition *STAGE20a* shows how we exit from our parallel flow. This transition will select the parent context and create the after stage *STAGE30* in our parent context. As there are no other active stages associated with our child context this ends our parallel workflow.

Decision Stage and Delay Stage

On completing a parallel workflow the decision stage *STAGE30* is used to check if there are any other parallel workflows in progress. If there are we move to a delay stage where the workflow waits for the other parallel flows to complete. On completing the other workflows will merge with the existing delay stage as it exists for the same business routine context. The transition *STAGE30a* will only create the delay stage if it does not already exist for the business routine context running the transition, hence it will only be created the first time the transition is run.

Once the final parallel workflow completes transition *STAGE30b* will be run. This uses a transition join which will pull down the delay stage and the manager stage and join them together with the decision stage back into a single path.

Exercises

1. Change transition *PLNAPPRVW2a* so that it also moves back onto stage *PLNAPPRVW2a*.
2. Add the delay stage *PLNAPPRVW5* to your BPD.
3. Add the decision stage *PLNAPPRVW4* to your BPD. This stage should have two cases defined below.
    1. The first case should check the value returned by the trainingmgr.xx_plan_app.count_assigned_reviewers(p_application_uref) function defined in your training package. If this is greater than 0 you should follow transition *PLNAPPRVW4a*.
    2. The second case should check the value returned by the trainingmgr.xx_plan_app.count_assigned_reviewers(p_application_uref) function defined in your training package. If this is equal to 0 then you should follow transition *PLNAPPRVW4b*.
4. Change transition *PLNAPPRVW3a* so that it now moves to after stage *PLNAPPRVW4* in the parent context.

Training:Architecture:Business Process Training Worksheets:Abort Pattern

Introduction

At times you may want to terminate the business process by ending all active stages. The following training sheet will show you how you can do this.

Concepts

A common requirement is to allow a user to terminate a business routine regardless of what stages it is currently in. For example an applicant may submit a request which has gone to be actioned which the applicant subsequently decides to cancel as he no longer requires the information. This request could be in one of many stages so we need to ensure that all of its active stages are ended in order to end the business routine and cancel the request.

In order to do this we provide an action to the user to cancel the routine. This action will raise a signal. Every stage which the business process could be in will implement a transition to end the stage that will be invoked on receiving the signal raised.

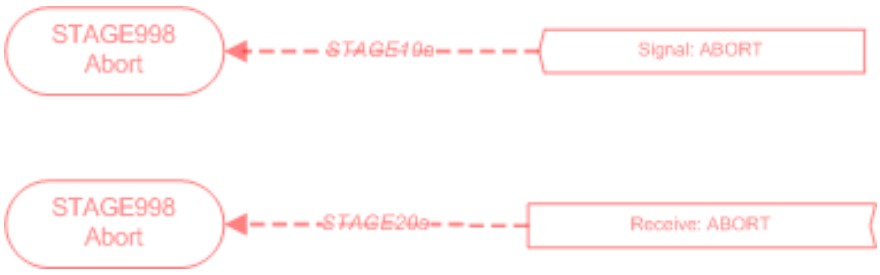
If you are aborting a subroutine the exit stage you move to can be a different exit stage from the exit stage entered when your routine completes successfully. This allows you to provide different feedback to your superroutine to inform it that the subroutine was aborted rather than completed should it need to take a different action. You need to be careful how you update the super-routine status if there are multiple stages being ended as a result of a signal. The reason for this is that if we are invoking a transition in the super-routine on its status being updated then after the first update the stage may no longer exist if it has transitioned to a different stage. Subsequent attempts to update its status will result in an application error. To get around this we can do one of two things:

1. Raise the signal in a *BEFORE_TRANSITION* block. This will prevent the super-routine from transitioning when the other stages are ended as we have not ended the before stage invoking the signal. This will only work if the status of the before stage is *ACTIVE*.
2. Have all transitions invoked by the signal move to an exit stage that does not update the super-routine status. Only the transition invoking the signal will move to an exit stage updating the super-routine status. This will ensure that the status is only updated once.

Abort Layer

To avoid cluttering your workflow you can mark your abort signals and stages as using the abort layer on your workflow diagram. The layer for a shape can be changed using the *Format Shape* toolbar. The advantage of using a special layer to on these shapes is that you can turn layers on and off in Visual Studio. This is done under the View -> Layer Properties menu option. By turning off the abort layer it may make it easier to understand the more important parts of a workflow diagram. Below is example markup for signal sends and receives marked up in the abort layer.

Workflow Abort Layer Markup



Exercises

1. Add a second exit stage to your subroutine *PLNAPPRVW998* which sets the superroutine status to *ABORTED*.
2. Add an action to *PLNAPPRVW2* with a prompt *Cancel Review* that runs transition *PLNAPPRVW2b* and raises the required signal. Provide a confirm dialog when the action is pressed.
3. Implement the transitions listening for your abort signal on stages *PLNAPPRVW3* and *PLNAPPRVW5*. For now on *PLNAPPRVW3* just move to stage *PLNAPPRVW998*, ignore stage *PLNAPPRVW6*.
4. Change your Planning Application business process so that on the status of stage *PLNAPP50* being updated to *ABORTED* it will run transition *PLNAPP50b*.

Training:Architecture:Business Process Training Worksheets:Orphan Stages

Introduction

This training sheet will explain orphan stages and how the are used in the business process.

Concepts

In general a routine has ended once all of its stages have ended. There may be times when you want a routine to end while it still has one or more stages active. A common scenario for this is when you want a super-routine to continue processing whilst still providing workbasket actions to one or more stages in its subroutine. As explained in the Subroutines training sheet a super-routine stage cannot end until all active stages on its subroutine have ended.

To get around this problem we can transition to an orphan stage. On transitioning to an orphan stage a new orphan business routine is created. The orphan routine is based on the same business process as its parent routine, however it does not start at an entry stage rather it immediately creates the orphan stage you are transitioning to. This has to be done in a new context as you cannot continue using the same business context from your parent routine. The orphan routine will create new business routine assignments which will be associated with the same assignments as the business context that created the orphan routine.

An orphan routine can be as simple or complicated as you like. Essentially it is like working in a new context except this context will allow its parent to end while it is still active. To create an orphan routine you invoke a transition with the after stage created in a new context. The new context is marked up with the *ORPHAN* attribute to show that the new context and stage should be created in an orphan routine.

BPD Markup for Orphan Transition

```
<TRANSITION>
  <TRANSITION_LABEL>STAGE10a</TRANSITION_LABEL>
  <MOVE>
```

```
    <BEFORE>
      <STAGE_LABEL>STAGE10</STAGE_LABEL>
    </BEFORE>
    <CONTEXT_SET>
      <NEW_CONTEXT_LIST>
        <CONTEXT_NAME>MY_ORPHAN_CONTEXT</CONTEXT_NAME>
      </NEW_CONTEXT_LIST>
      <ORPHAN>true</ORPHAN>
      <AFTER>
        <STAGE_LABEL>STAGE20</STAGE_LABEL>
      </AFTER>
    </CONTEXT_SET>
  </MOVE>
</TRANSITION>
```

Workflow Markup for Orphan Transition

Exercises

1. Create stage *PLNAPPRVW6*. This stage should have a single action available to your *REVIEWER* assignment labelled *Clear* which will run transition *PLNAPPRVW6a*.
2. Modify transition *PLNAPPRVW3b* so that it forks to stages *PLNAPPRVW998* and *PLNAPPRVW6*. Stage *PLNAPPRVW6* should be created as an orphan stage.

Training:Architecture:Business Process Training Worksheets:Conditional Action Sets

Introduction

The following section will explain how to conditionally provide actions to users on a business process stage.

Concepts

Assignments control what actions sets are available to what workbaskets on a stage. At times though you may want to provide an action set to a workbasket only if certain conditions have been met, otherwise the action set should not be created. For each action set you can optionally provide a list of conditions that must be met in order for the action set to be created on entering a stage.

The conditions you define on an action set can take any of *QUERY_NAME|WHEN|AND|OR|NOT|NO* elements as defined in the Run API training sheet. On moving to a stage if an action set has any conditions attached to it these will be evaluated. If they evaluate to true the action set will be created, if they evaluate to false the action set will not be created.

BPD Markup for Conditional Action Sets

```
<ACTION_SET>
  <ACTION_SET_MNEM>i</ACTION_SET_MNEM>
  <CONDITION_LIST>
    <QUERY_NAME>Check My Condition</QUERY_NAME>
  </CONDITION_LIST>
  <WORKBASKET_ASSIGNMENT_LIST>
    ...
  </WORKBASKET_ASSIGNMENT_LIST>
  <ACTION_LIST>
    ..
  </ACTION_LIST>
</ACTION_SET>
```

Exercises

1. Currently we can keep trying to assign more reviewers on our manager stage. If there are no further reviewers to accept these reviews this will result in an application error. To fix this add a condition to the assign reviewer action so that it is only created if unassigned reviewers exist. You can use the function trainingmgr.xx_plan_app.count_unassigned_reviewers(p_application_uref) to get the number of unassigned reviewers (replacing *xx* with your initials).

Training:Architecture:Business Process Training Worksheets:Action Overlay

Introduction

This section will show you how you can add additional descriptive information to an action set.

Concepts

On defining a stage you can provide it with an *ACTION_DESC_STAGE* element. This is static text that will be displayed in the workbasket against all action sets for the stage. You can define further static information for an action set using the *ACTION_DESC_ACTION* element in the *ACTION_DATA* for an action set.

If you want to show additional dynamic information against an action set this can be done by providing a query that returns this information and running the query in your action set. The query you run must return XML in the format shown below:

```
SELECT
  XMLELEMENT("ROOT"
  , XMLELEMENT("ACTION_DESC_ACTION", 'Some more information')
  )
FROM dual
```

Within your action set you use an *ACTION_OVERLAY_LIST* element to define the additional information to be displayed. The information will be appended as a comma separated list to the existing action description. In the workbasket actions are grouped by their action description. If you provide static or dynamic action descriptions against an action set making its description different from other action sets then its actions will be displayed separate from the other action sets actions.

BPD Markup for Action Overlay

```
<ACTION_SET>
  <ACTION_SET_MNEM>i</ACTION_SET_MNEM>
  <WORKBASKET_ASSIGNMENT_LIST>
    ...
  </WORKBASKET_ASSIGNMENT_LIST>
  <ACTION_LIST>
    ...
  </ACTION_LIST>
  <ACTION_DATA>
    ...
  </ACTION_DATA>
  <ACTION_OVERLAY_LIST>
    <QUERY_NAME>My Action Overlay Query</QUERY_NAME>
  </ACTION_OVERLAY_LIST>
</ACTION_SET>
```

You can either call a query defined in your query definitions as shown in the above example or you can define the query directly inside an *SQL* element.

Exercises

1. Add a query to your query definitions to retrieve action description XML showing how many reviews are currently assigned and how many more reviewers are available. You can use the function trainingmgr.xx_plan_app.review_manage_action_desc(p_application_id) to retrieve a string with this information (replacing xx with your initials).
2. Add an action overlay to both your *Cancel Review* action on stage *PLNAPPRVW2*. The action overlay should use your previously defined query.
3. View how your actions are set-out (you may need to churn the stage by assigning a reviewers or cancelling and restarting the review). Notice how the *Assign Reviewer* action and *Cancel Review* action are shown against their different action descriptions.
4. Add the same action overlay to the *Assign Reviewer* action. Churn the stage to view the difference in how the actions are displayed.

Training:Architecture:Business Process Training Worksheets:Refreshing Stage Actions

Introduction

The following training sheet will show you how to recreate the actions on a stage when that stage already exists.

Concepts

Excluding context changes there are three ways in which you can move to a new stage, each of which has different results. We can move to a new stage that does not already exist, move from a stage to a different stage that already exists, or move from a stage to that same stage again.

Moving to a New Stage

When moving to a new stage the stage is created and the stage's action sets are created as described in the Moving Between Stages training sheet.

Moving to an Existing Stage

When moving from a stage to a different stage that already exists then the before stage will be ended as expected and the after stage will not be changed in any way.

Moving Onto the Same Stage

When moving from a stage back onto the same stage again then the stage will not be ended and hence will not be created again. What will happen though is that the action sets on the stage will be refreshed. This means that the stage actions will be ended and recreated. This is useful for when you want to update the actions on a stage for example in the case of conditional actions, changes to workbasket assignments, or dynamic action descriptions.

We have seen examples of this being used when we take ownership (see Taking Ownership training worksheet), and when we recreate our manager stage after assigning a reviewer in transition *PLNAPPRVW2a*. In this instance the condition is being evaluated each time controlling whether or not we see the *Assign Reviewer* action and the action description is being updated.

Exercises

In our review we want the manager to be able to assign a review again to a reviewer who has completed a review. In order to do this we need to update the action sets on the manager stage when a review is complete.

1. Change transition *PLNAPPRVW4a* so that it raises the signal *CHURN_PNAPPRVW2*.
2. Add transition *PLNAPPRWV2c* that will be invoked on receiving your *CHURN_PNAPPRVW2* signal.

Training:Architecture:Business Process Training Worksheets:On Stage Tasks

Introduction

This section will introduce how you can carry out a set of tasks each time a stage is created.

Concepts

As part of the markup for a stage you can specify certain tasks to perform when a stage is created. These tasks are similar to what you can do in a transition. The advantage of marking them up on the stage rather than the transition to a stage is that you may enter a stage from many different transitions. If every time the stage is created you need to perform the same tasks you can reduce code duplication by marking up the tasks to perform on the stage itself rather than on all the transitions into it.

BPD Markup for Stage Tasks

```
<STAGE>
  <STAGE_LABEL>STAGE10</STAGE_LABEL>
  <STAGE_TITLE>Sample Stage Tasks</STAGE_TITLE>
  <ON_STAGE>
    <API_LIST>
      <QUERY_NAME>Do Stuff</QUERY_NAME>
    </API_LIST>
  </ON_STAGE>
  <ACTION_LIST>
    ...
  </ACTION_LIST>
</STAGE>
```

Note: Any tasks specified in the *ON_STAGE* element will be run everytime the stage is created. Ensure that this is the behaviour you require. If transitioning from the stage back onto itself the *ON_STAGE* tasks will NOT be run as the stage already existed.

Training:Architecture:Business Process Training Worksheets:Using Process XML Packages

Introduction

This section will show you how to easily use the functions defined in your process XML package.

Concepts

We have seen how we can provide functions scoped to our UREF in the Process XML training worksheet. We have also seen a few examples of how the business process makes use of this by providing information in the workbasket, header information for your modules, and subject text for emails. You can also put any of your own functions in your process XML package and access them easily through the *bpmmgr.uref* package without the need to write dynamic SQL.

The *bpmmgr.uref* package provides three useful functions for accessing the correct process XML package for a given function. These are getProcessedXML, getProcessedNumber, and getProcessedString which return XML, a number, and a string respectively. You must provide the function with your UREF and the name of the function you want to call. You may also optionally provide XML with additional parameters.

```
l_result_id := bpmmgr.uref.getProcessedNumber(
  p_Uref => l_my_uref
, p_Result_Type => 'RETURN_MY_NUMBER'
);
```

To be supported by the functions in the UREF package the function you define in your process XML package must accept the ID of the case it is handling and XML parameters even if it does not use them. Below is a sample specification of a function that can be used with getProcessedNumber.

```
FUNCTION return_my_number (
  p_key IN VARCHAR2
, p_xml_params IN XMLTYPE DEFAULT NULL
) RETURN NUMBER;
```

Exercises

In our review process we have several calls to the trainingmgr.xx_plan_app package. The ties our subroutine to our Planning Application rather than it being generic as planned. To get around this problem we need to move these functions to our process XML package. This will allow us to use our generic review process with any application so long as it defines the required functions in its process XML package.

1. Move the get_unassigned_reviewer, count_assigned_reviewers, count_unassigned_reviewers, and review_manage_action_desc functions to your process XML package. Note that you will need to change them slightly so that they accept an ID rather than a UREF (ensure you change the body as well) and they need an additional parameter for the XML parameter.
2. Alter your review business process so that it makes use of the functions provided in your process XML package by calling the appropriate UREF function.