Workflow
SQL for Workflow Training Workbook
September 2007
Release 4.4

SUNGARD HIGHER EDUCATION

What can we help you achieve?

SunGard Higher Education

4 Country View Road
Malvern, Pennsylvania 19355
United States of America
(800) 522 - 4827

Customer Support Center website
http://connect.sungardhe.com

Distribution Services e-mail address

distserv@sungardhe.com

Other services

In preparing and providing this publication, SunGard Higher Education is not rendering legal, accounting, or other similar professional services. SunGard Higher Education makes no claims that an institution's use of this publication or the software for which it is provided will insure compliance with applicable federal or state laws, rules, or regulations. Each organization should seek legal, accounting and other similar professional services from competent providers of the organization's own choosing.

Trademark

Notice of rights

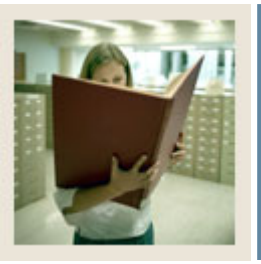


Table of Contents

Table of Contents (Continued)

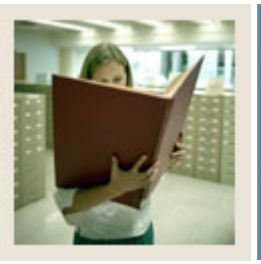


Table of Contents (Continued)

Section A: Introduction

Lesson: Overview

Jump to TOC

Workbook goal

The goal of the *SQL for Workflow* workbook is to provide you with the knowledge and skill to build Oracle database objects to integrate with the Workflow Modeler. The workbook is divided into these sections:

- Introduction
- Creating Events
- Defining Business Components
- Testing and Error Handling
- Creating Database Packages
- Workflow Migration
- Bonus Material

Intended audience

Technical individuals who need to understand how to automatically start a workflow and how to select or update data used in the workflow model using two of SunGard Higher Education's products, Banner and Workflow. This includes, but is not limited to, system analysts, database administrators, and technical/development managers who

- are familiar with Banner and Workflow
- have familiarity with PL/SQL scripting
- have relational database experience.

Objectives

On completion of this session, you should be able to

- state how Workflow events and business components connect to Oracle objects
- describe Workflow schemas and grants
- describe event handling tables, triggers, and procedures needed to start a workflow (using Workflow)
- understand how to select or update data used in the Workflow modeler
- state how Workflow handles data types and null values
- discuss best practices for naming conventions and defining Oracle schemas.

Section A: Introduction

Lesson: Overview (Continued)

Jump to TOC

Prerequisites

The following Workflow training sessions should have been completed.

- Workflow Technical Training
- Workflow Process Modeling Training

Section contents

Section A: Introduction

Lesson: Understanding the Big Picture

Jump to TOC

Introduction

This training focuses on how to make two software applications, Banner and Workflow, "talk" to each other and thereby start an automated workflow or retrieve and update data from the Banner tables.

Note: This session does not cover how to setup Workflow to "talk" to any other third-party application in order to start an automated workflow.

How Workflow works

An *automated* workflow

1. starts with an **event** that causes a change to an Oracle table in Banner.
2. a **database trigger** logs that event to the Banner Event tables. (These tables are constantly polled by the Workflow Event Dispatcher, alerting Workflow to any table changes that have occurred and indicating whether a model should be started.)
3. procedures retrieve or update data from the Banner database and are attached to workflow activities via **business components**.
4. pre-defined **component activities** and decision trees in the Workflow modeler help "route" information to the correct roles and open appropriate Banner forms or applications.

Applying a real-world scenario

Throughout this training, an address change scenario will be used to illustrate the procedures and processes associated with an automated workflow. This scenario forms the basis for practice opportunities, as well.

Address Change Scenario – Pat Nelson is an engineering major at your institution. His family recently moved from Pennsylvania to Texas. Prior to the move, Pat entered an address change into the university's system via Self Service.

Section A: Introduction

Lesson: Understanding the Big Picture (Continued)

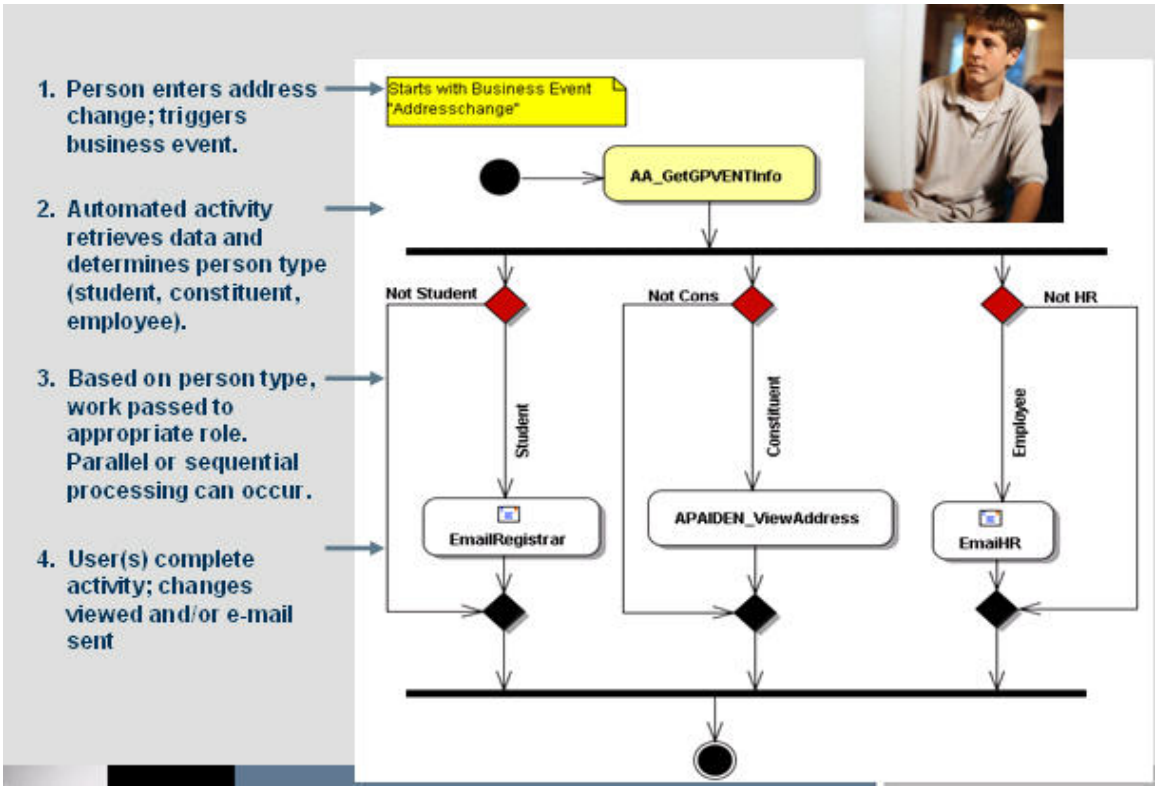
Jump to TOC

Applying a real-world scenario, continued

Applying this address change scenario to the automated workflow process described previously results in the following…

- The **event** is Pat entering his address change in Banner Self Service.
- The **database trigger** records the appropriate information about the address change from the Oracle tables and logs the information to the Banner event tables.
- **Business components** in Workflow are set up such that the data is interpreted (e.g., is the address change associated to a student, employee, or constituent?) and routed to the appropriate role (e.g., in the case of a student the registrar would want to know about the address change).
- A **component activity**, set up in Workflow, sends a **notification** to a person in the Registrar's office who can then click on the notification and be taken directly to the appropriate Banner form for updating purposes.

…and looks like this.



Section A: Introduction

Lesson: Understanding the Big Picture (Continued)

Jump to TOC

Oracle objects used by Workflow

To make Workflow and Banner talk to each other as described, various Oracle objects are used. For example:

- To start a workflow model, you use either **triggers** or **procedures** to populate Banner event tables.
- To retrieve and/or update data in Oracle tables you use either **procedures** or **packages**.
- To call a function, you use **Workflow sqlQuery business component**.

All three types of Oracle Database objects will be discussed in this training workbook.

Storing Oracle objects

Database objects (e.g., triggers, procedures, packages, and queries) are stored in a schema. Some common Workflow schemas are listed in the following table. You will use WFOBJECTS to store the database objects created during this training.

Name	Description
WORKFLOW	Stores all Workflow model and instance data
WFBANNER	Exchanges data with Banner when extracting Banner work items
WFEVENT	Polls events from the Banner event table.
WFSSO	Provides single sign-on for Banner and Workflow
WFAUTO	**Grants execute access to stored procedures and queries**
WFOBJECTS	**Stores Workflow triggers, procedures, and packages**

Section A: Introduction

Lesson: Terminology

Jump to TOC

Event

External action or database change that can start a workflow automatically. This is done by updating the event tables in Banner generally through a database trigger or a script. Event setup must be completed in both Banner and Workflow for the event to fire. In Workflow, this term is called a Business Event.

Note: Banner Events with a target system of Workflow link to a Business Event in Workflow with the same name.

Business component

An external executable object (form, procedure, sqlQuery) to be launched by a Workflow component activity.

Database trigger

Procedural code that is automatically executed in response to certain actions on a particular table in a database.

Function

Procedural code stored in the database that performs a specific task. SQL functions typically return only one data value.

Oracle database objects

Include packages, procedures, functions, triggers, tables, views

Packages

A group of procedures, functions, and sql code used to store related objects.

Stored Procedures

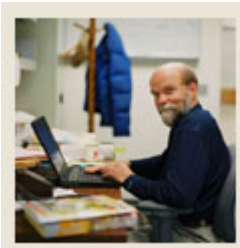
Procedural code stored in the database to access or update the database. Procedures can return a set of data values. Procedures are available to other applications requiring data base access.

Schema

The database objects a user owns are collectively called a schema. The delivered schema for storing Workflow objects is wfobjects.

Target system

The system using the Banner event records, for example Workflow and Luminis.



Section B: Creating Events

Lesson: Overview

Jump to TOC

Purpose

In this section, you will learn the steps involved in creating events, both for Banner and Workflow. You will also have the opportunity to practice creating events.

Introduction

An **event** is something that will eventually launch an automated workflow. Events must be created in both Banner and in Workflow before a workflow can automatically launch.

There are five steps to creating an event:

- Create an event in Banner.
- Write a trigger to initiate an event in Banner.
- Define a Workflow model.
- Create a Workflow business event.
- Create the business process in Workflow.

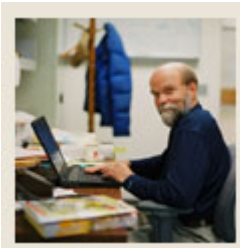
You can complete these steps in any order; however, all steps must be completed and they all need to be coordinated with each other. The **event name** is the common thread that runs through all steps and links them together.

Tip: Defining the Banner event prior to defining the Workflow event will allow you to use the Event Wizard in Workflow to define the Workflow event. For more information on the Event Wizard, refer to the *Workflow Integration User Manual*.

Objectives

On completion of this section, you should be able to

- describe common Workflow schemas
- create an event in Banner
- write a trigger to initiate an event in Banner
- define a Workflow model
- create a Workflow business event
- create the business process in Workflow
- state trigger grants
- describe restrictions on triggers
- create trigger code
- drop or disable a trigger.

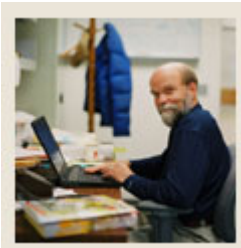


Section B: Creating Events

Lesson: Overview (Continued)

Jump to TOC

Section contents



Section B: Creating Events

Lesson: Creating an Event in Banner

Jump to TOC

Purpose

In this lesson, you will learn how to create an event in Banner. You will also have the opportunity to practice creating an event in Banner using an address change scenario.

Introduction

An event is something that will eventually launch a workflow. When you define a Banner event, you are setting up Banner tables to "accept" a business event. You might want to capture a business event in Banner tables when

- there is a change to a table in the Banner database
- specific conditions are met in PL/SLQ code.

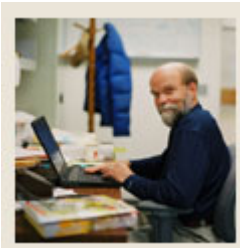
For Banner to "accept" events, the following validation and rule forms need to be completed.

Validation/Rule Forms	Purpose
Target System Code Validation Form (GTVEQTS)	Ensures the target system is Workflow.
Event Queue Code Validation Form (GTVEQNM)	Create the event name
Parameter Group Code Validation Form (GTVEQPG)	Create the parameter group name
Parameter Code Validation Form (GTVEQPM)	Select all parameters for the event
Parameter Group Definitions Form (GOREQPG)	Associate parameters to the parameter group name
Event Queue Name Definitions Form (GOREQNM)	Associate the event name to the parameter group name

By completing these forms you will

- ensure the target system is Workflow
- create the event name
- create the parameter group name
- select all (or enter new) parameters for the event
- associate parameters to the parameter group name
- associate the event name to the parameter group name

Note: These forms are found in the **Banner General** module under **System Functions / Administration**. They are in the **Event Queue Maintenance** folder.



Section B: Creating Events

Lesson: Exercise 1 – Create an Event in Banner

Jump to TOC

Exercise 1 – Create an event in Banner

Address Change Scenario – Pat Nelson is an engineering major at your institution. His family recently moved from Pennsylvania to Texas. Prior to the move, Pat entered an address change into the university's system via Self Service. Your institution wants to set up a workflow to automate the handling of an address change request.

1. Use the following forms to create a Banner event for the address change scenario described.

- Target System Code Validation Form (GTVEQTS)
- Event Queue Code Validation Form (GTVEQNM)
- Parameter Group Code Validation Form (GTVEQPG)
- Parameter Code Validation Form (GTVEQPM)
- Parameter Group Definitions Form (GOREQPG)
- Event Queue Name Definitions Form (GOREQNM)

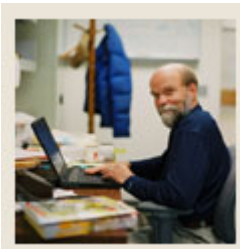
Additional information

2. Name the event ADDRESSCHANGExx (where *xx* is your assigned training account number).

3. Name the parameter group ADDRCH*xx* (where *xx* is your assigned training account number).

4. The following parameters (in this order) are needed for the event.

    1. EVENTNAME –ADDRESSCHANGE*xx*
    2. PRODUCTTYPE – SCT Banner
    3. WORKFLOWSPECIFICNAME
    4. PIDM
    5. ID
    6. FullName
    7. AddressType
    8. UserName

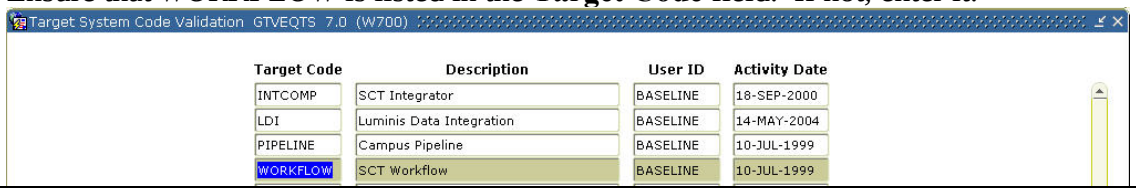
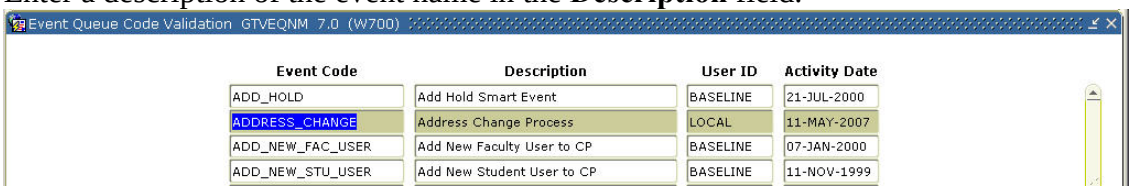


Section B: Creating Events

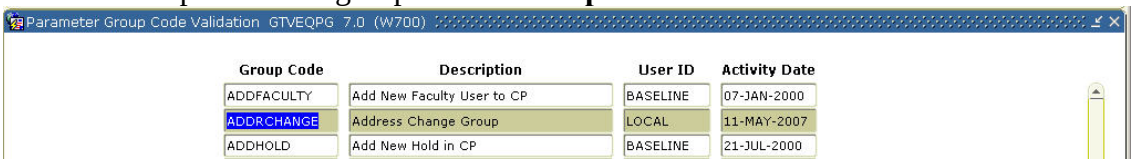
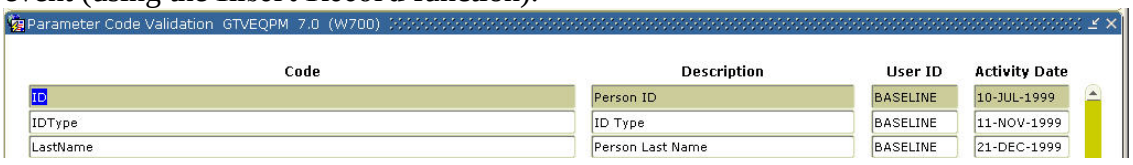
Lesson: Exercise 1 – Solution

Jump to TOC

Solution

Step	Action
Ensure target system is Workflow	
1	Access the Target System Code Validation Form (GTVEQTS).
2	Ensure that *WORKFLOW* is listed in the **Target Code** field. If not, enter it. 
Create the event name	
	Note: You can access the Event Queue Code Validation Form (GTVEQNM) while on the Target System Code Validation Form (GTVEQTS). Click on the **Options** menu and select *Add/Review Event Codes (GTVEQNM)*.
3	Access the Event Queue Code Validation Form (GTVEQNM). Note: You might want to review the event codes to ensure the event is not already listed.
4	Perform an **Insert Record** function.
5	Enter a name for the event in the **Event Code** field (20 character limit). Example: ADDRESSCHANGE.
6	Enter a description of the event name in the **Description** field.  Note: The **User ID** field will default to *LOCAL*. The **Activity Date** will default to today's date.
7	Click the **Save** icon.

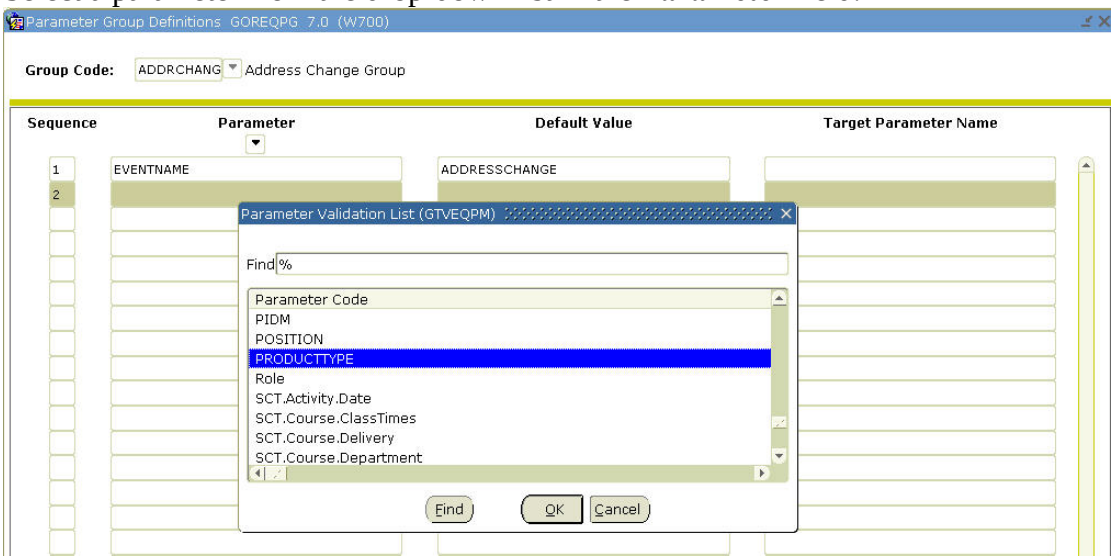
Solution, continued

Step	Action
Create the parameter group name	
	Note: You can access the Parameter Group Code Validation Form (GTVEQPG) while on the Event Queue Code Validation Form (GTVEQNM). Click on the **Options** menu and select *Add/Review Parameter Groups (GTVEQPG).*
8	Perform an **Insert Record** function.
9	Enter a group code name in the **Group Code** field (10 character limit). Note: All parameters for the event will be grouped under this name and ultimately passed to Workflow. It is important to make the group code name meaningful.
10	Enter a description of the group in the **Description** field.  Parameter Group Code Validation GTVEQPG 7.0 (W700) Group Code / Description / User ID / Activity Date ADDFACULTY / Add New Faculty User to CP / BASELINE / 07-JAN-2000 ADDRCHANGE / Address Change Group / LOCAL / 11-MAY-2007 ADDHOLD / Add New Hold in CP / BASELINE / 21-JUL-2000 Note: The **User ID** field will default to *LOCAL*. The **Activity Date** will default to today's date.
11	Click the **Save** icon.
Select all (or enter new) parameter names	
	Note: You can access the Parameter Code Validation Form (GTVEQPM) while on the Parameter Group Code Validation Form (GTVEQPG). Click on the **Options** menu and select *Add/Review Parameter Names (GTVEQPM).*
12	Review the list to determine whether there is an existing parameter name for each parameter in the event. If not, enter a new parameter name for each parameter in the event (using the **Insert Record** function).  Parameter Code Validation GTVEQPM 7.0 (W700) Code / Description / User ID / Activity Date ID / Person ID / BASELINE / 10-JUL-1999 IDType / ID Type / BASELINE / 11-NOV-1999 LastName / Person Last Name / BASELINE / 21-DEC-1999 Example: Address change parameters might include ID, Fullname, User, etc. Note: Parameter names do not have to match. They are simply place holders for the data from the database trigger.

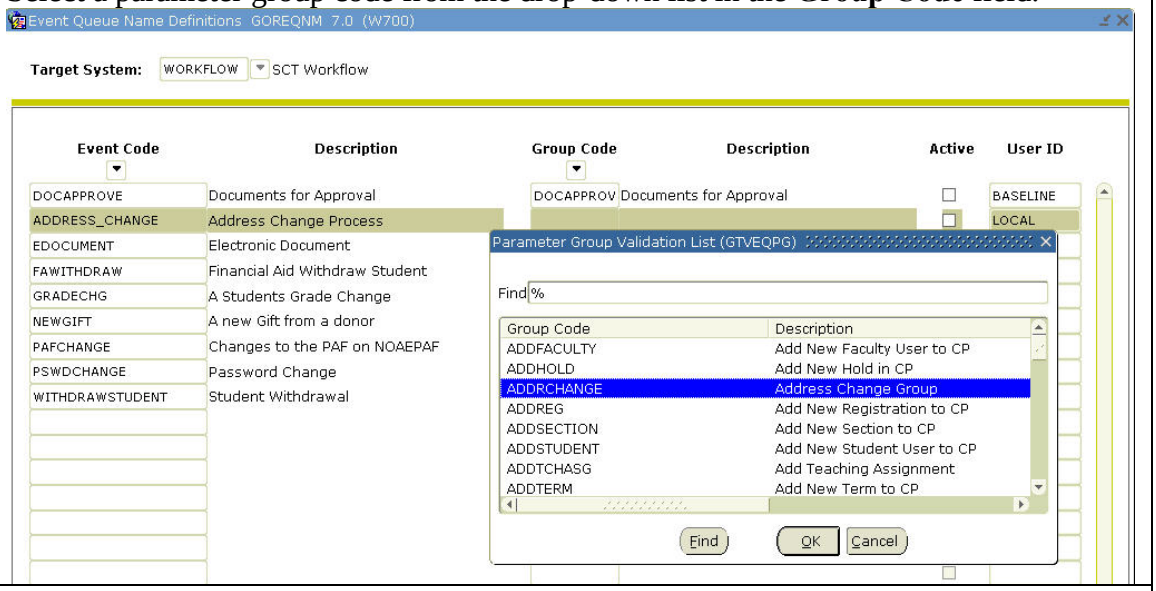
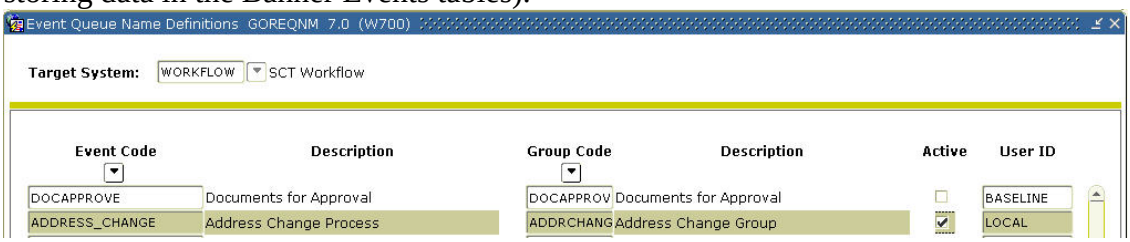
Solution, continued

Step	Action
Associate parameters to the parameter group name	
	Note: You can access the Parameter Group Definitions Form (GOREQPG) while on the Parameter Code Validation Form (GTVEQPM). Click on the **Options** menu and select *Add/Review Parameter Group Rules (GOREQPG).*
13	Enter the group code name in the **Group Code** field. Note: The group code name is the same name you entered on the Parameter Group Code Validation Form (GTVEQPG).
14	Perform a **Next Block** function.
15	Enter a number in the **Sequence** field. Note: All parameters that the event will use must be listed here. The parameters must be listed on this form in the same order as in the code (trigger, script, program).

Solution, continued

Step	Action
16	Select a parameter from the drop-down list in the **Parameter** field.  Note: The first three parameters must be • EVENTNAME – This is the name of the event added to the GTVEQNM table and supplied by the trigger. • PRODUCTTYPE • WORKFLOWSPECIFICNAME – This is similar to the subject line of an email. It appears as the first column on the Workflow Worklist and is passed from the trigger. (e.g. New Applicant – Mary Smith)
17	Enter the default name in the **Default Value** field. Note: PRODUCTTYPE must be *SCT Banner* (to access Banner data).
18	Repeat Steps 15-17 for the remaining parameters associated with the event. 4. PIDM 5. ID 6. FullName 7. AddressType 8. UserName
19	Click the **Save** icon.

Solution, continued

Step	Action
Associate the event name to the parameter group name	
	Note: You can access the Event Queue Name Definitions Form (GOREQNM) while on the Parameter Group Definitions Form (GOREQPG). Click on the **Options** menu and select *Add/Review Event Definitions (GOREQNM)*.
20	Enter or select the target system from the drop-down list in the **Target System** field.
21	Perform a **Next Block** function.
22	Perform an Insert Record
23	Select an event code from the drop-down list in the **Event Code** field.
24	Select a parameter group code from the drop-down list in the **Group Code** field. 
25	Click the **Active** checkbox to make the event active (i.e., you want the event to start storing data in the Banner Events tables).  Note: The workflow will start only if the event is active.
26	Click the **Save** icon.

Section B: Creating Events

Lesson: Writing a Database Trigger

Jump to TOC

Purpose

There are five steps to creating an event:

- Create an event in Banner.
- Write a trigger to initiate an event in Banner.
- Define a Workflow model.
- Create a business event in Workflow.
- Create the business process in Workflow.

In the previous lesson, you created an event in Banner. In this lesson, you will learn how to write a trigger to initiate the Banner event you created. You will also have the opportunity to practice writing a database trigger for the address change event and test it.

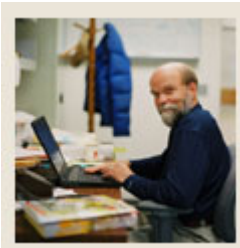
About database triggers

A database trigger is procedural code that is automatically executed in response to certain actions on a particular table in a database. Database triggers for Workflow are written in much the same way as a non-Workflow trigger.

Database triggers for Workflow are typically stored in the Workflow database, WFOBJECTS. They can fire before or after inserts, updates, and deletes are made to a table. Database triggers can fire for each row or each statement (at the table level).

To start a workflow, a database trigger must populate two Banner tables

- Event Queue Base Table (GOBEQRC)
- Event Queue Repeating Table (GOREQRC)



Section B: Creating Events

Lesson: Writing a Database Trigger (Continued)

Jump to TOC

Introduction

There are three tasks that the database trigger must perform.

1. Ensure Workflow is enabled at your institution.
2. Ensure the event is defined with the target system of Workflow.
3. Pass the data from the trigger (vparms) and insert it into the Banner Event tables (GOBEQRC and GOREQRC).

SunGard Higher Education delivers three event packages which should be incorporated into your trigger. These packages are stored in the BANINST1 schema. They can be called from third-party applications. The three event packages are

- goksyst (ensures that Workflow is enabled)
- gokevent (ensures that the event is defined with the target system of Workflow)
- gokparm (passes data from the trigger [vparms] and inserts it into the Banner Event tables)

The gokparm package also updates the Banner Events tables so that Workflow knows there is work waiting. The event name identified in the trigger *must match* the Banner event name and the Workflow event name. As mentioned earlier, the event name creates the link between Banner, the trigger, and Workflow.

Section B: Creating Events

Lesson: Writing a Database Trigger (Continued)

Jump to TOC

Event trigger code

Code must be included in the trigger to check whether Workflow is enabled and to check for the existence of the Banner event. In addition code must be written to pass parameter information and insert it into Banner event tables. What follows is the actual code for each of the packages.

goksyst (to check for Workflow)
If goksyst.f_isSystemLinkEnabled('WORKFLOW') THEN

gokevent (to check for existence of event)
:=substr(gokevnt.F_CheckEvent
('WORKFLOW',‘ADDRESSCHANGE'),1,20);

Note: The event name in the trigger code *must match* the event name created on GTVEQNM.

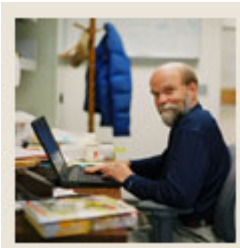
gokparm (to insert parameter data into event tables)
v_params(1).param_value := ‘ADDRESSCHANGE';
v_params(2).param_value := 'SCT Banner';
v_params(3).param_value := Address Change for '|| full_name ||’ ID-’ ||id;
v_params(4).param_value := pidm;
v_params(5).param_value := full_name;
v_params(6).param_value := id;
v_params(7).param_value:=.new.spraddr_atyp_code
v_params(8).param_value := username;
gokparm.Send_Param_List(event_code, v_Params);

Note: The first three parameters are required. The remaining parameters represent data needed by Workflow to start a workflow.

v_params(1) is the event name. This should match the name in the GTVEQNM table.
v_params(2) is the Product type. It specifies that the event is connected to the Banner Database. This field is case sensitive and MUST be spelled “SCT Banner.”
v_params(3) is the Workflow specific name. For clarity, ensure the name includes specific data to identify what this workflow instance is about (e.g., Address Change – Mary Smith).

Privileges

In order for triggers to work, trigger grants need to be created. Privileges also need to be set up on any tables being accessed in the trigger (e.g., SPRIDEN or STVNATN).



Row-level triggers

Row-level triggers fire once per row and are processed by the trigger statement. Within the trigger, you may access the row currently being processed by referencing pseudo-records :old and :new.

Example: v_Parms(4).param_value := :NEW.spraddr_pidm

When clause

When clauses are valid for row-level triggers only. They are used to restrict when a trigger fires. Colons in front of new and old pseudo-columns are only used within a trigger body. You can omit colons in front of old and new pseudo-columns within the When clause.

Example:

```
CREATE OR REPLACE TRIGGER st_spraddr_addresschange
   AFTER INSERT OF spraddr_addresschange ON spraddr
   FOR EACH ROW
   WHEN (new.spraddr_seqno>1 BEGIN
…
END;
```

Trigger restrictions

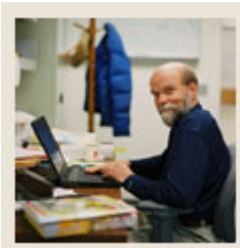
You may not issue transactional statements, such as COMMIT, ROLLBACK or SAVEPOINT on triggers. Procedures called from a trigger CANNOT contain any transaction statements on the same table or view containing tables that the trigger is on.

Example:

```
Select SPRADDR_STREET_LINE1 from SPRADDR
where SPRADDR_ATYP_CODE='MA';
```

(Issues a "mutating table" error.)

The trigger body CANNOT declare any LONG or LONG RAW variables. The :new and :old CANNOT refer to a LONG or LONG RAW column.



Viewing

Query the Oracle database view ALL_TRIGGERS to view stored trigger codes.

To view trigger errors, select from view ALL_ERRORS if the errors occur while trigger is created.

Query database views, USER_TRIGGERS (under own schema) or ALL_TRIGGERS (for any trigger that schema has access to) to view trigger code.

Removing or disabling triggers

To remove a trigger, use the following syntax:

```
DROP TRIGGER trigger_name;
```

To temporarily disable a trigger, use the following syntax:

```
ALTER TRIGGER trigger_name [DISABLE | ENABLE];
```

Order of trigger firing

Execute before statement-level trigger, if present. For each row affected by the statement

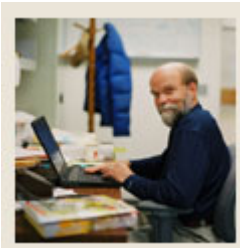
- Execute BEFORE row-level trigger, if present
- Execute statement
- Execute AFTER row-level trigger, if present

Execute after statement-level trigger, if present.

Note: If there are multiple triggers of the same type on a table, there is no way to tell in what order the triggers will fire.

Triggers and rollbacks

If a record is rolled back, the row-level trigger will NOT fire. Event tables are populated only when the record is committed. The model does NOT start.



Section B: Creating Events

Lesson: Exercise 2 – Write a Database Trigger

Jump to TOC

Exercise 2 – Write a database trigger

1. Create a trigger on the SPRADDR table to fire when an address has been changed. Use the following syntax when creating the trigger.

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER } triggering_event ON table_reference
[FOR EACH ROW [WHEN trigger_condition ]] trigger_body;
```

Additional information

2. Name the trigger, ST_SPRADDR_AddressChangexx (where *xx* is your assigned training number).

3. Include a check for Workflow (goksyst)

4. Include a check for existence of event (gokevent)

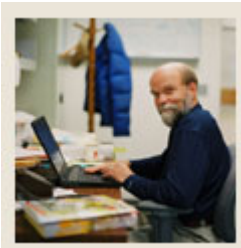
5. Include exactly the same parameters as those listed in GOREQPG and ensure they are in the same order (gokparm)

6. Test the trigger by inactivating an old address in Banner and adding a new address (on SPAIDEN).

   Note: Testing procedures follow.

7. Your trigger should have created an event in Banner on the GOBEQRC and GOREQRC tables. Using GOAEQRM, review whether a new record was created.

Best Practice: The process for changing an address in Banner is to make the current address inactive and insert a new address. That way, a history of address changes will be logged.



Section B: Creating Events

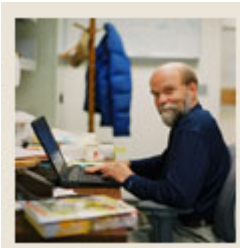
Lesson: Exercise 2 – Write a Database Trigger (Continued)

Jump to TOC

Testing a trigger

Follow these steps to test your trigger.

Step	Action
Inactivating an address and inserting a new one	
1	Log in to the Banner instance that is integrated with Workflow. Note: You must log in as *wfuserxx* (where *xx* is your assigned training number) and use the password: *u_pick_it*. If you do not, the trigger will not create your event properly.
2	Enter SPAIDEN in the **Go To** field.
3	Enter an ID number in the **ID** field.
4	Perform a **Next Block** function.
5	Click on the **Address** tab.
6	Click on the **Inactive** checkbox of any current address record.
7	Click the **Insert Record** icon. Type the same address as the one you just inactivated.
8	Click the **Save** icon. Then, close the form.
9	Enter GOAEQRM in the **Go To** field.
10	Enter *Workflow* in the **Target System Code** field.
11	Enter *ADDRESSCHANGExx* in the **Event Code** field.
12	Select *None* from the drop-down menu in the **Status** field.
13	Perform a **Next Block** function.
14	Verify the event has a status of *processed*. Note: This means the Event Dispatcher picked up this event.
15	Make note of the sequence number for the address change you just created. Write the sequence number in the space provided. Seq: _____ Note: No Workflow models are started because no Workflow model, event, and business process have been set up.



Section B: Creating Events

Lesson: Exercise 2 – Write a Database Trigger (Continued)

Jump to TOC

Testing a trigger, continued

Step	Action
Testing the trigger in Workflow	
16	Log into Workflow. Note: You must log in as *wfuserxx* (where *xx* is your assigned training number) and use the password: *u_pick_it*.
17	Click *Business Events* in the **Administration** block.
18	Click on **External Events**. Note: You'll see *"failed"* (default) in the Search window.
19	Click the **Search** button.
20	All the events that were processed by the Event Dispatcher but had no Workflow business event to match are listed here.

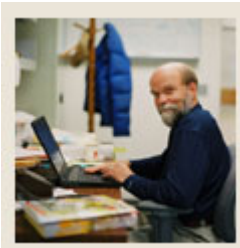
Section B: Creating Events

Lesson: Exercise 2 – Solution

Jump to TOC

Solution

```
Create or Replace TRIGGER ST_SPRADDR_AddressChangexx
-- FILE NAME..: ST_SPRADDR_AddressChangexx.sql
-- BANNER REL.: 7X
-- OBJECT NAME: ST_SPRADDR_AddressChangexx
-- PRODUCT....: BANNER GENERAL
-- USAGE......: Post to Banner Workflow Event Tables GOREQRC, QOBEQRC
AFTER INSERT ON spraddr
FOR EACH ROW
   WHEN (NEW.spraddr_seqno>1) AND user='WFUSERxx'
DECLARE

      v_Params          Gokparm.t_parameterlist;
      event_code        gtveqnm.gtveqnm_code%TYPE;
      fullname          VARCHAR2(50);
      p_id              spriden.spriden_id%TYPE;

BEGIN

      IF  Goksyst.f_isSystemLinkEnabled('WORKFLOW')
                  THEN

                  -- Check for the event definition and set the event code.
      event_code
:=SUBSTR(Gokevnt.F_CheckEvent('WORKFLOW','ADDRESSCHANGExx'),1,20);
      IF event_code <> 'NULL' then
      SELECT spriden_id, spriden_first_name || ' ' || spriden_last_name INTO
p_id, fullname FROM spriden WHERE spriden_change_ind IS NULL AND
spriden_pidm=:NEW.spraddr_pidm;
```



Solution, continued

```
BEGIN


            --  Set the parameter values
            v_Params(1).param_value := 'ADDRESSCHANGExx';
            v_Params(2).param_value := 'SCT Banner';
            v_Params(3).param_value := 'Address CHANGE FOR - ' || fullname ||
' ID - '||p_id;
            v_Params(4).param_value := :NEW.spraddr_pidm;
            v_Params(5).param_value := fullname;
            v_Params(6).param_value := p_id;
            v_Params(7).param_value:=.new.spraddr_atyp_code
            v_Params(8).param_value := :NEW.spraddr_user;

            -- Create the event
            Gokparm.Send_Param_List(event_code, v_Params);

            END IF;
      END IF;
END;
```

Section B: Creating Events

Lesson: Defining a Workflow Model and Associated Event and Process

Jump to TOC

Purpose

There are five steps to creating an event:

- Create an event in Banner.
- Write a trigger to initiate an event in Banner.
- Define a Workflow model.
- Create a business event in Workflow.
- Create the business process in Workflow.

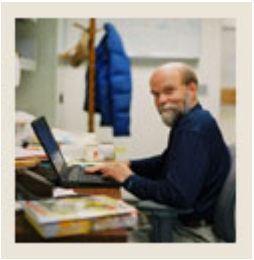
In the previous lessons, you created a Banner event and wrote a trigger. In this lesson, you will learn how to define a Workflow model, create a business event, and create the business process.

Defining a Workflow model

The steps involved in defining a model follow.

1. Enter a workflow definition
2. Create a diagram
3. Define activity properties
4. Validate the model and change status
5. Test the model by manually starting the workflow and completing the activity(ies)

Note: These steps are covered in detail in the *Workflow Process Modeling Training Workbook* (Release 4.3).



Section B: Creating Events

Lesson: Defining a Workflow Model and Associated Event and Process (Continued)

Jump to TOC

Creating a business event

It is important to understand the difference between a business event and a business process.

In Workflow, a business event accepts parameter data from Banner to pass to the Workflow model. In other words, a *business event* in Workflow is used to indicate *what* event will start a Workflow model. A *business process* in Workflow determines *when* the model will start, based on criteria that you define.

Workflow Component	Action
Workflow Business Event	*What* data does the event pass to the model?
Workflow Business Process	*When* does the model start?

How does it work?

For everything to work accurately, the event name in Banner must match exactly with the name of the business event in Workflow. Parameters are passed from the trigger to Banner and then from Banner to Workflow. The event parameters set up in the trigger must be mapped to *required* context parameters in Workflow.

Parameter values can be passed (or connected) to more than one Workflow model. In other words, one business event can start many models.

Creating the business process

A Workflow business process determines *when* a Workflow model will start. For example, does a Workflow model start every single time a business event fires? Or, are there conditions (criteria; called guard conditions in Workflow) under which the model starts?

Ultimately, the Workflow model, the business event, and the business process are connected (or associated) to each other. A business process may be associated with only one model at any given time.

Section B: Creating Events

Lesson: Exercise 3 – Define a Workflow Model

Jump to TOC

Exercise 3 – Define a Workflow Model

1. Create a workflow model with one manual activity to display the data parameters passed from the event.

    - Name the manual activity, DisplayAddressChangexx (where *xx* is your assigned training number).

2. Using the Event Wizard, create a Workflow business event for ADDRESSCHANGExx.

    - Associate the workflow model with the workflow event.

3. Create a business process and associate the workflow model and event with the business process. Add two guard conditions:

    - ADDRTYPE = “MA”
    - UserName = WFUSERxx (where *xx* is your assigned training number).

4. Test the event and model.

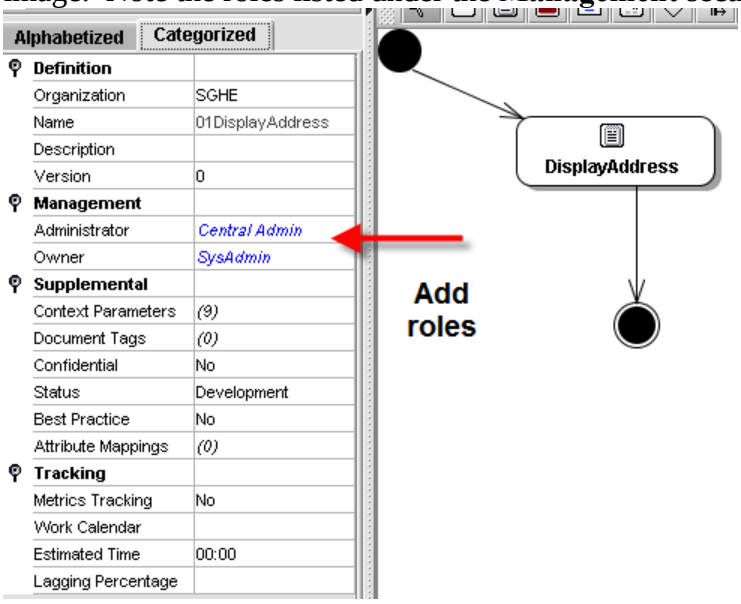
Section B: Creating Events

Lesson: Exercise 3 – Solution

Jump to TOC

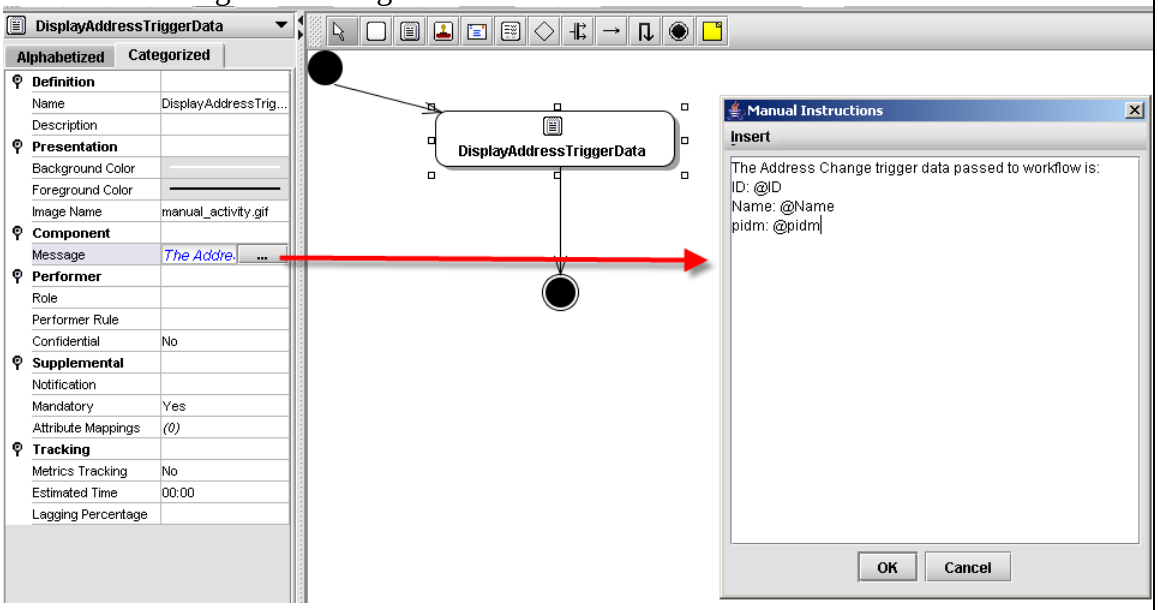
Solution

Step	Action
Create a Workflow Model	
	Note: The property sheet for the Workflow model should look like the following screen image. Note the roles listed under the **Management** section.
	Note: The following context parameters should be included on the property sheet.

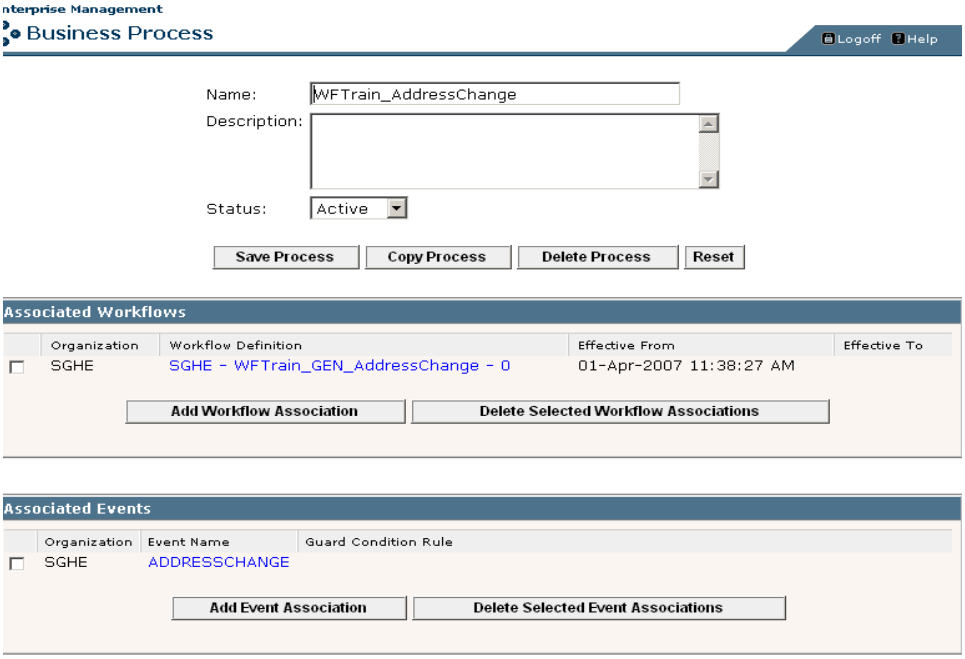


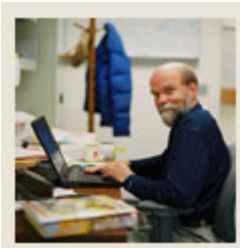
Name	Type	Required
Name	Text	No
ID	Text	No
PIDM	Numeric	No

Solution, continued

Step	Action
	Note: The property sheet for the manual activity, *xxDisplayAddressEvent*, should look like the following screen image. 
Create a Workflow business event	
1	Select *Business Events* from the **Administration** tab.
2	Click on **Business Event Definitions**.
3	Click the **Event Wizard**. Note: If you do not use the **Event Wizard** and chose to enter the parameters manually, see the section, *Entering Parameters Manually*, at the end of this procedure.
4	Choose *Workflow* from the drop-down list.
5	Select your Banner event (*ADDRESSCHANGExx*) from the drop-down list.
6	Click the **Create Event** button.
7	Click the **Save** icon.

Solution, continued

Step	Action
Associate the event to the model	
1	Scroll down to the **Associated Workflows** tab.
2	Click the **Add Workflow Association** button.
3	Select your Workflow business event (*ADDRESSCHANGExx*) from the drop-down list.
4	Click the **Save** icon.
5	Map the event parameters (Banner) to the context parameters (Workflow) as shown below.  Note: The parameters may appear in a different order.



Solution, continued

Step	Action
Create a business process	
1	Click on *Business Processes* in the **Enterprise Managemen**t window.
2	Scroll down and click the **Add Business Process** button.
3	Enter *AddressChangeProcessxx* in the **Name** field.
4	Enter *Process to notify departments when MA address change made in Banner* in the **Description** field.
5	Select *Active* from the drop-down list in the **Status** field.
6	Click the **Save Process** button.
7	Click **Save Association**.
Associate an event to the process	
1	Scroll to the **Associated Events** tab.
2	Click **Add Event Association**.
3	Select the *ADDRESSCHANGExx* event form the drop-down list.
4	Click **Save Association**.
Associate business process to model	
1	Scroll to the **Associated Workflows** tab.
2	Click on the workflow name.
3	Click in the **Effective To** field to enter an end date and time. Note: Include note about can associate more than one process to a model but only one can be active at one time.
4	Click **Save Association**.
5	Click **Add Workflow Association**.
6	Select the *ADDRESSCHANGExx* workflow from the drop-down list. Note: The current date will default.
7	Click **Save Association**.

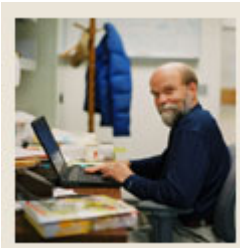
Solution, continued

Adding a guard condition

Workflow business processes can contain guard conditions (criteria) for starting a model.

Note: For purposes of this training session, you will add a guard condition to your business process so that the workflow will start only if your Banner User ID is the initiating User ID that adds the address change to the Banner database. If this guard condition wasn't added, the first person to make an address change would initiate everyone's workflows.

Step	Action
Add a guard condition to your business process	
1	Click on the *AddressChangeXX* event in **Associated Events** tab.
2	Double-click on ADDRTYPE in the **Events Parameters** column. Note: This will paste the parameter name in the guard condition rule block.
3	Double-click the equal sign (=) in the **Operators** column. Then, enter *MA* in upper case letters with double quotes. Example: ADDRTYPE = "MA"
4	Double-click on *USERID* in the **Events Parameters** column. Note: This will paste the parameter name in the guard condition rule block.
5	Double-click the equal sign (=) in the **Operators** column. Then, enter your Banner User ID in upper case letters with double quotes. Example: USERID = "WFUSERxx" (where *xx* is your assigned training number).
6	Click **Save Association**.



Solution, continued

Step	Action
Inactivate an address and insert a new one	
1	Log in to the Banner instance that is integrated with Workflow. Note: You must log in as *wfuserxx* (where *xx* is your assigned training number) and use the password: *u_pick_it*. If you do not, the trigger will not create your event properly.
2	Enter SPAIDEN in the **Go To** field.
3	Enter an ID number in the **ID** field.
4	Perform a **Next Block** function.
5	Click on the **Address** tab.
6	Click on the **Inactive** checkbox of any current address record.
7	Click the **Insert Record** icon. Type the same address as the one you just inactivated.
8	Click the **Save** icon. Then, close the form.
9	Enter GOAEQRM in the **Go To** field.
10	Enter *Workflow* in the **Target System Code** field.
11	Enter *ADDRESSCHANGExx* in the **Event Code** field.
12	Select *None* from the drop-down menu in the **Status** field.
13	Perform a **Next Block** function.
14	Verify the event has a status of *processed*. Note: This means the Workflow Event Dispatcher picked up this event.
15	Make note of the sequence number for the address change you just created. Write the sequence number in the space provided. Seq: _____

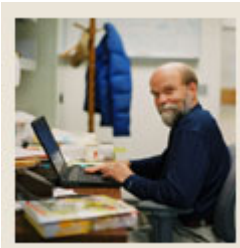
Solution, continued

Step	Action
Simulate the business event	
1	Select *Business Events* from the **Administration** tab.
2	Click on **Business Events Definitions**.
3	Select your event, *ADDRESSCHANGExx*.
4	Click the **Simulate Event** button.
5	Enter *ADDRESSCHANGExx TEST* in the **Workflow Specifics Name** field.
6	Enter the appropriate values in the **Business Event Parameters** fields. Note: If you don't have the necessary values, use the **Workflow Status Search** to review a previous workflow run and use the same parameter values.
7	Click the **Post Event** button.
8	Click **View** to see the posted event.

Entering parameters manually

In some cases, you may want to enter parameters manually versus using the Event Wizard in Workflow. What follows are step-by-step instructions for entering parameters manually.

Step	Action
1	Select *Business Events* from the **Administration** tab.
2	Click on *Business Event Definitions*.
3	Enter parameters manually. Note: Context parameters in Workflow must match exactly with Banner event parameter names on GOREQPG. Names are case-sensitive; however they can be entered in any order. In addition, they must match the **Target Field** name in Banner. If the **Target Field** name is blank, the names must match exactly with those used in the **Parameter Name** field. Note: The product type must match exactly with Banner Parameter 2, SCT Banner. Names are case-sensitive.
4	Click the **Save** icon.



Section B: Creating Events

Lesson: Handling External Events

Jump to TOC

Introduction

At runtime, the Event Dispatcher posts the event in Workflow and schedules it for execution. Workflow then evaluates the event and marks its status as either

- Failed (failed evaluation)
- Completed (successfully completed evaluation)
- Pending (scheduled and awaiting processing)
- Initial (posted but not yet scheduled)

When evaluating the event, Workflow checks if the event is associated with any business processes. Then, for each business process it checks

- the status of the business process
- any guard conditions
- the current time stamp.

External events and the evaluation process can be viewed in Workflow.

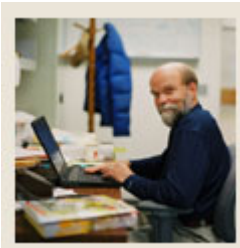
Section B: Creating Events

Lesson: Exercise 4 – Determine External Event Status

Jump to TOC

Exercise 4 – Determine External Event Status

1. Find the event in the External Event window of Workflow.
2. Note the status.



Section B: Creating Events

Lesson: Exercise 4 – Solution

Jump to TOC

Solution

Step	Action
Determine external event status	
1	Log into Workflow. Note: You must log in as *wfuserxx* (where *xx* is your assigned training number) and use the password: *u_pick_it*.
2	Click *Business Events* in the **Administration** block.
3	Click on **External Events**. Enter *AddressChangexx* in the Event Name window. Note: The Status block indicates whether the event was Completed of Failed. A blank will display every event, regardless of status.
4	Click the **Search** button.
5	All the events that were processed by the Event Dispatcher are listed here.

Section B: Creating Events

Lesson: Applying Constraints

Jump to TOC

Introduction

There are a number of options for placing constraints on a Workflow model. When doing so, it is important to remember the following.

- Place the constraint in the trigger row logic if the condition applies to ALL models. For example:

```
FOR EACH ROW
WHEN new.spraddr_atyp_code='MA'
```

- Place in trigger body if the condition requires data from additional tables and is applicable to ALL models.
- Place in the business process if the condition applies to ONE model.
- Place in the model if logic is required to perform particular model activities.

Section B: Creating Events

Lesson: Exercise 5 – Place Constraints

Jump to TOC

Exercise 5 – Place constraints

1. Determine where to place constraints for each of the following:

    - Address Change process: If the student has financial aid, certain activities must be performed by the financial aid office.

    - Gift Stewardship process: Only notify the gift stewards if the gift amount is more than $5,000.

Section B: Creating Events

Lesson: Exercise 5 – Solution

Jump to TOC

Solution

There are really no *wrong* answers to this question. It all depends.

- **Address Change process** – If the student has financial aid, certain activities must be performed by the financial aid office.

  Option 1: Add code to the trigger.
  You may want to place code directly in the trigger to pass an indicator as to whether the student has financial aid. Then you could also set up a guard condition in the business process to start the financial aid model only if the financial aid indicator is true.

  Option 2: Check for financial aid in the model
  Add a component activity as the first step of the model to check if the student has financial aid.

- **Gift Stewardship process** – Only notify the gift stewards if the gift amount is more than $5,000.

  Option 1: Create a guard condition in the model.
  If the model incorporates various paths for the different gift amounts, it would be very easy to add a guard condition to check the gift amount.

  Option 2: Create a guard condition in the business process.
  If the gift stewardship process is complex enough to be it's own model, it may be better to add a guard condition directly to the business process to start the model.

If you have other ideas, please discuss them and write your thoughts here.

Section B: Creating Events

Lesson: Revising a Business Event

Jump to TOC

Purpose

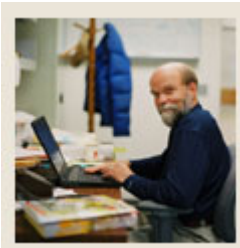
In previous lessons, you learned how to create a Banner event and a Workflow business event. In some cases, you may find that you need to revise a Workflow business event. In this lesson, you will learn how to do just that

Introduction

Once an event has been logged, changes can no longer be made to the group parameters in GOREQPG. You will need to complete the following steps to revise a business event in Workflow.

- Delete data in GOBEQRC and GOREQRC for the Banner event using SQLplus (Tables are connected by seqno.)
- Delete event record from GOREQNM
- Make modifications to GOREQPG
- Add event record to GOREQNM
- Make necessary changes to the Workflow business event.

Note: If you make changes in your non-production environment, be sure to make the same changes in the production environment after you migrate the workflow.



Section B: Creating Events

Lesson: Exercise 6 – Delete Event Records

Jump to TOC

Exercise 6 – Delete event records

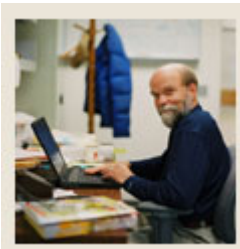
1. Delete your AddressChangexx records from the Banner tables (GOBERQC and GOREQRC).

2. Delete event record from GOREQNM.

3. Make modifications to GOREQPG.

4. Add event record to GOREQNM.

5. Make necessary changes to the Workflow business event.



Section B: Creating Events

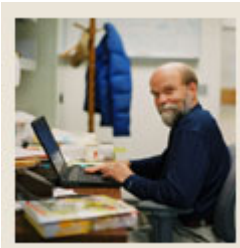
Lesson: Exercise 6 – Solution

Jump to TOC

Solution

Step	Action
Delete data in Banner event tables (GOBEQRC and GOREQRC)	
1	Access GOBEQRC. Note: This form is the base header table for the event. The sequence in this parent table is associated with the records in the child table (GOREQRC).
2	Find your event records. Note the value of GOBEQRC_SEQNO. SQLplus code: `Select * from GOBEQRC where GOBEQRC_EQNM_CODE='01ADDRESSCHANGE'.`

3	Delete all records from both tables having the same sequence number.
	
	SQLplus code:
	`delete from GOBEQRC where GOBEQRC_SEQNO=#;`
	`delete from GOREQRC where GOREQRC_SEQNO=#;`
	
	`Alternatative code`
	delete from GOREQRC where GOREQRC_SEQ_NO in
	(select GOBEQRC_SEQNO from GOBEQRC where GOBEQRC_EQNM_CODE='EVENT_NAME');



Solution, continued

Step	Action
Delete event record in Banner	
4	Access GOREQNM.
5	Delete the record association the event and the group.
Make modifications to the event definition in Banner	
6	Access GOREQPG.
7	Make the necessary changes.
Add event record in Banner	
8	Access GOREQNM.
9	Re-associate the event and group code.
Make changes to Workflow business event	
10	Select *Business Events* from the **Workflow Administration** tab.
11	Click on the *Business Event Definitions* option.
12	Make changes to the business event definition *manually* (if the Event Wizard has already been used to establish the business event in Workflow). Note: Ensure any parameter changes in Banner match exactly to Workflow parameters.
13	Ensure the model is re-associated with the event and with the appropriate parameter mappings. Note: You can click on the workflow name in the **Business Event** window to review and/or change mappings.
14	Ensure that the workflow and the event are associated with the business process in the **Event Management** tab.
15	Add any necessary guard conditions for the event association.
Activate the event in Banner	
16	Access GOREQNM.
17	Check the **Active** checkbox.
18	Click the **Save** icon.

Section B: Creating Events

Lesson: Writing a Procedure

Jump to TOC

Purpose

In this lesson, you will learn how to initiate a Banner event by creating a procedure to call another model from a workflow.

Introduction

When data changes in a Banner table, you can write a trigger to initiate a Banner event. If you want to initiate a Banner event when there are no data changes occurring to the Banner tables, then you can initiate a Banner event by

- creating a procedure to call another model from a workflow
- creating a batch process.

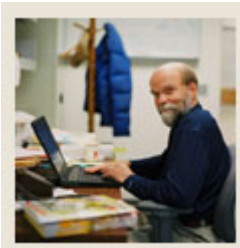
Information on how to create a batch process to initiate a Banner event will be covered in the next lesson.

Creating a procedure that calls another model from a workflow

There may be times when you'd like to launch a second workflow from an activity in a workflow. Take, for example, the Address Change Scenario that was introduced earlier. Pat, who is an engineering student moving from Pennsylvania to Texas, has been awarded a financial aid package. When out-of-state moves occur, it would be helpful if another workflow started from the Address Change workflow. This second workflow would alert the Financial Aid Department to changes in state residence.

In this case, you cannot write a trigger because there is no Banner table change to monitor. However, you can launch a second workflow from a workflow activity by

- creating a procedure that populates the Banner event tables
- attaching a business component (for the procedure) to the workflow activity.



Section B: Creating Events

Lesson: Writing a Procedure (Continued)

Jump to TOC

Writing a procedure

The logic for writing a procedure is exactly the same as writing a trigger.

Action	Package
Ensure Workflow is enabled.	goksyst
Ensure the event is defined with the target system of Workflow.	gokevent
Pass data from the trigger [vparms] and insert it into the Banner Event tables.	gokparm

Section B: Creating Events

Lesson: Exercise 7 – Write a Procedure

Jump to TOC

Exercise 7 – Write a Procedure

1. Write a procedure that populates Banner event tables for the financial aid “out-of-state” process.

Additional Information

2. Name your procedure P_FA_STATE_CHANGExx (where xx is your assigned training number)

3. Include check for Workflow (goksyst)

4. Include check for existence of event (gokevnt)

5. Parameters passed to procedure from the Workflow model = fullname and ID. Parameters passed to Workflow = FullName and ID (gokparm)

Section B: Creating Events

Lesson: Exercise 7 – Solution

Jump to TOC

Solution

```
CREATE OR REPLACE PROCEDURE P_FA_STATE_CHANGExx(pidm IN NUMBER,
full_name IN VARCHAR2, p_id IN VARCHAR2) IS

  v_Params                    Gokparm.t_parameterlist;
  event_code                  gtveqnm.gtveqnm_code%TYPE;

BEGIN

  IF  Goksyst.f_isSystemLinkEnabled('WORKFLOW') THEN
     event_code
:=SUBSTR(Gokevnt.F_CheckEvent('WORKFLOW','FA_STATE_CHANGE'),1,20);
      IF event_code <> 'NULL' then
      --  pass parameters to the event
      v_Params(1).param_value := 'FA_STATE_CHANGE';
      v_Params(2).param_value := 'SCT Banner';
      v_Params(3).param_value := 'FA State Change for '|| full_name ||'
'||id;
      v_Params(4).param_value := pidm;
      v_Params(5).param_value := p_id;
      v_params(6).param_value := full_name;
      Gokparm.Send_Param_List(event_code, v_Params);

  END IF;
END IF;
END;
```

Section B: Creating Events

Lesson: Creating a Batch Process

Jump to TOC

Purpose

In this lesson, you will learn how to create a batch process to initiate a Banner event.

Creating a batch process

The first way to initiate a Banner event is to create a procedure that calls another model from a workflow. The second way to initiate a Banner event is to create a batch procedure. For example, the Advancement Office would like to start a process when ALL addresses are flagged as inactive (I).

In this case, you cannot write a trigger because additional records in the same table (SPRADDR) must be checked to ensure they have an inactive status (mutating table). However, you can

- create a batch process with a cursor to check if all records with a "MA" address type have a status indicator (on SPRADDR) of *inactive* and
- populate the Banner event tables
- run the procedure with a cron or AppWorx job.

Section B: Creating Events

Lesson: Exercise 8 – Create a Batch Process to Start a Model

Jump to TOC

Exercise 8 – Create a batch process to start a model

1. Write a procedure to check whether address records for the address type “MA” are inactive.

Additional information

2. Check SPRADDR.
3. Limit your check to records changed since yesterday at midnight.
4. If all addresses are inactive, populate the Banner event tables.
5. Parameters passed to procedure from the Workflow model = PIDM, ID, and FullName.

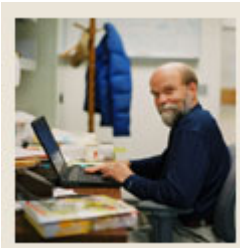
Section B: Creating Events

Lesson: Exercise 8 – Solution

Jump to TOC

Solution

```
CREATE OR REPLACE PROCEDURE wfobjects.P_ADDRESSES_INACTIVExx
IS

  v_Params              Gokparm.t_parameterlist;
  event_code            gtveqnm.gtveqnm_code%type;
  v_id                  spriden.spriden_id%type;
  v_fullname            varchar2(40);
  v_pidm                spriden.spriden_pidm%type;

cursor c_address_inactive is
SELECT spraddr_pidm FROM spraddr a WHERE
(SELECT count(b.spraddr_pidm) FROM spraddr b WHERE
a.spraddr_pidm=b.spraddr_pidm
AND b.spraddr_status_ind IS NULL AND b.SPRADDR_ATYP_CODE='MA')=0

AND SPRADDR_ATYP_CODE='MA';
--AND SPRADDR_ACTIVITY_DATE > TO_DATE(TO_CHAR(SYSDATE-1,'MM-DD-YYYY'),'MM-DD-
YYYY');
```



Solution, continued

```
BEGIN

  IF  Goksyst.f_isSystemLinkEnabled('WORKFLOW') THEN
     event_code :=SUBSTR(Gokevnt.F_CheckEvent('WORKFLOW',
'INACTVADDRESS'),1,20);
    IF event_code <>'NULL' then

    open c_address_inactive;
    loop
    fetch c_address_inactive into v_pidm;
    exit when c_address_inactive%notfound;

    select spriden_id, spriden_first_name || ' '||spriden_last_name into
v_id, v_fullname
        from spriden where spriden_pidm=v_pidm and spriden_change_ind is
null;

      --  pass parameters to the event
      v_Params(1).param_value := event_code;
      v_Params(2).param_value := 'SCT Banner';
      v_Params(3).param_value := 'MA Address Inactive or Missing for '||
v_fullname ||' '||v_id;
      v_Params(4).param_value := v_pidm;
      v_Params(5).param_value := v_fullname;
      v_params(6).param_value := v_id;
      Gokparm.Send_Param_List(event_code, v_Params);
   end loop;

      END IF;
    END IF;
END;
```

Section B: Creating Events

Lesson: Troubleshooting Events

Jump to TOC

Troubleshooting Banner events

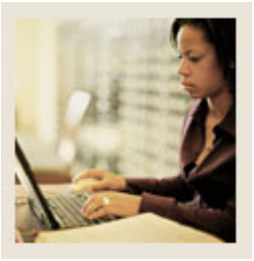
If a Workflow model does not start, check the following in Banner.

- ❑ Is the event record in GOREQMN active?
- ❑ Are trigger parameters the same as banner event parameters in GOREQPG?
- ❑ Is the event logged in GOAEQRM?
- ❑ Is the event on GOAEQRM shown as "*Processed*"?
- ❑ On GOREQPG, is SCT Banner in Parameter 2 spelled correctly and using the correct case? (It is case-sensitive.)

Troubleshooting Workflow business events

If a Workflow model does not start, check the following in Workflow.

- ❑ Is the event in External Events? If not, check that the Event Dispatcher is running. The configuration.xml event flag must be enabled. Also, check the status of failed events.
- ❑ Check if the Workflow business event is associated with model.
- ❑ Check if the business process is associated with the workflow.
- ❑ Check the guard condition on the business process.



Section C: Defining Business Components

Lesson: Overview

Jump to TOC

Introduction

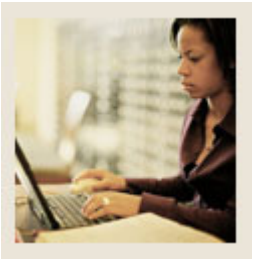
In this section, you will gain a better understanding of business components and how workflow activities pass data through them. You will also have the opportunity to practice each step and apply what you have learned.

Objectives

On completion of this session, you should be able to

- define a business component
- explain how workflow activities get data through business components.

Section contents



Section C: Defining Business Components

Lesson: Understanding Business Components

Jump to TOC

Introduction

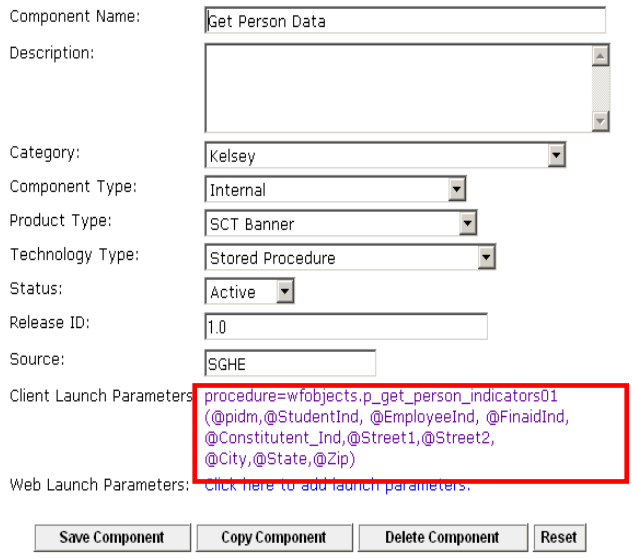
Business components define the work that will be performed by an external entity such as

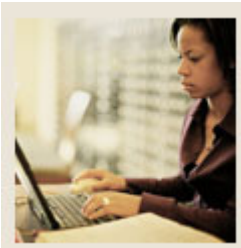
- launch a Banner form
- launch a desktop application
- run a SQL procedure
- run a SQL query.

Relationship between components, activities, and procedures and queries

Procedures and sql queries are defined in business components. Procedures and sql queries do the work of selecting and updating data. They

- select data from a database source(s) and pass it into a workflow (e.g., email=student@univ.edu).
- check table values and pass indicators to a workflow (e.g., student=yes)
- update database records (e.g., GURMAIL).





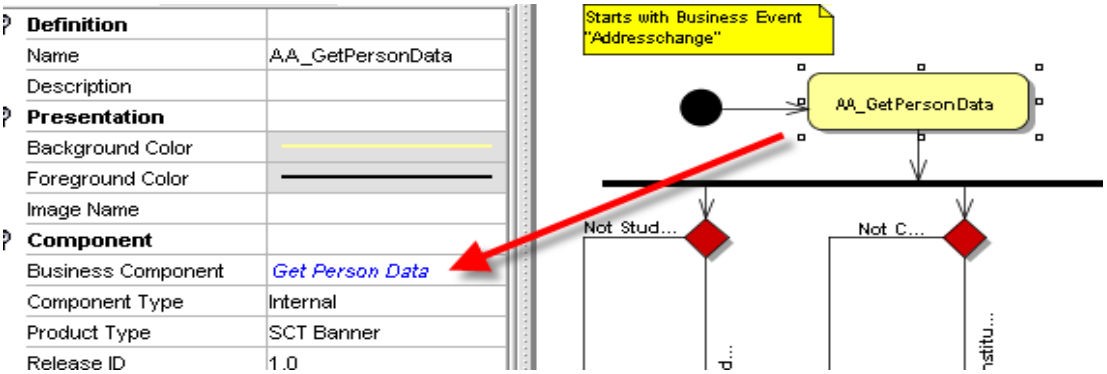
Section C: Defining Business Components

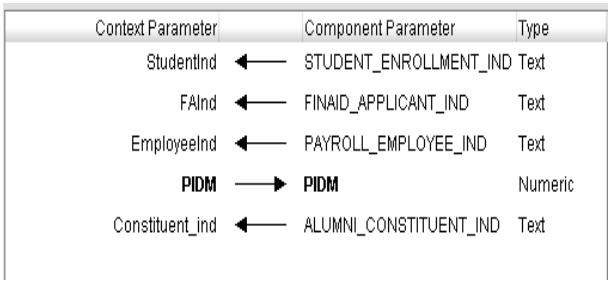
Lesson: Understanding Business Components (Continued)

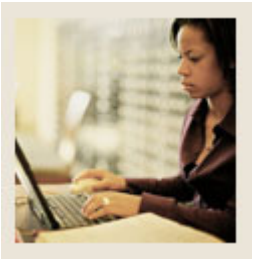
Jump to TOC

Relationship between components, activities, and procedures and queries, continued

For each activity in a workflow, a business component is identified and the data that the workflow needs (e.g., context and component parameters) is defined.







Section C: Defining Business Components

Lesson: Understanding Business Components (Continued)

Jump to TOC

Business component attributes

Business components consist of a

- component type
- product type
- technology type.

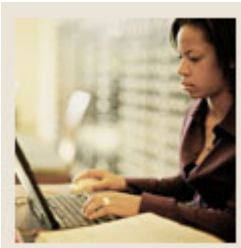
The **component type** indicates the type of program a business component represents and where it should be executed. Component types include

- Interactive (user input is required)
- Internal (no user interaction is required)
- Automated Workflow aware (no user interaction is required; third-party application)
- Automated non-Workflow aware (no user interaction is required; third party application).

The **product type** identifies the product associated with the business component (e.g., SCT Banner, Desktop).

The technology type informs Workflow how an external application should be launched. For example, should the application be launched via

- Desktop application
- Banner
- Workflow Aware Desktop Application component
- sqlQuery
- Stored procedure
- Web application



Section C: Defining Business Components

Lesson: Understanding Business Components (Continued)

Jump to TOC

Internal component types

Of the four component types, the focus of this training will be on internal component types. There are two internal component types

- Oracle procedure, which runs a SQL procedure against a data source defined in the product type (e.g., Banner, Desktop).
- sqlQuery, which runs a sqlQuery against a data source defined in the product type.

Business component parameters

For the procedures and queries to do the work of selecting and updating data, they must know what data is needed by the workflow to complete the activity.

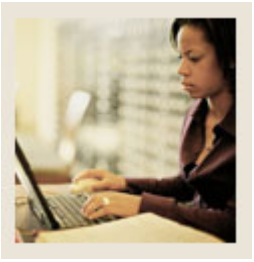
Parameters are the pieces of data passed to and from the business component.

- **R**equired **IN**put – data passed *in* to the Oracle object from the Workflow model. Needed to start a procedure or a SQL query.
- **G**uaranteed **O**utput – data passed *out* to the Workflow model from the Oracle object. Needed for Workflow to continue processing.

Remember **RINGO**!

Parameters are defined on the business component. They are referred to as launch parameters and are used for data communication between business components and Workflow. They must begin with an asterisk @ and match component parameters in spelling and case.

Technology Type	**Launch Parameter Name**	**Parameter Example**
SQL Query	**sqlQuery**	select spriden_id into **@ID** from spriden where spriden_change_ind is null and spriden_pidm=@pidm
Stored Procedure	**procedure**	wfpkg_goremail.get_email_address **(@pidm,@emailAddr)**



Section C: Defining Business Components

Lesson: Understanding sqlQueries

Jump to TOC

Introduction

In this lesson, you will learn about the internal business components that call SQL queries and their advantages and disadvantages. You will also learn how to construct sqlQuery, understand grants needed for sqlQuery tables, and learn how to use this type of internal business component in a Workflow activity.

sqlQueries

SQL queries

- select data from a database source(s) and pass into workflow
- check table values and pass indicators to workflow
- update database records.

The use of sqlQuery affords the following advantages and disadvantages.

Advantages	Disadvantages
• Easy to write • Used for simple database queries • No Oracle access needed. Can be written in Workflow.	• No inherent error checking • Error will raise Workflow alert and Workflow model will stop • Will return values from only one row • Can only SELECT; not UPDATE • Code limit of 255 characters

sqlQuery parameters

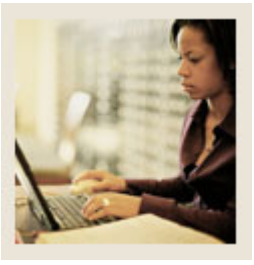
Parameter names in SQL query must match component parameter names. Parameter names are case sensitive.

Example: organization ≠ Organization; sqlQuery ≠ sqlquery

sqlQuery grants

At runtime, Workflow uses WFAUTO to run sqlQueries. Select privileges for any tables used in a SELECT statement must be granted to the WFAUTO schema.

Example: `sql>grant select on spriden to wfauto;`



Section C: Defining Business Components

Lesson: Exercise 9 – Create a sqlQuery Business Component

Jump to TOC

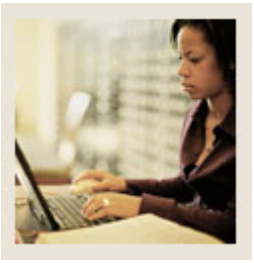
Exercise 9 – Create a sqlQuery

1. Create a sqlQuery business component to retrieve the current full name from SPRIDEN.

   - Input parameter = ID
   - Output parameter = FullName

2. Create a Workflow model to call the business component and display the ID and FullName to the screen.

3. Validate and run the model.



Section C: Defining Business Components

Lesson: Exercise 9 – Solution

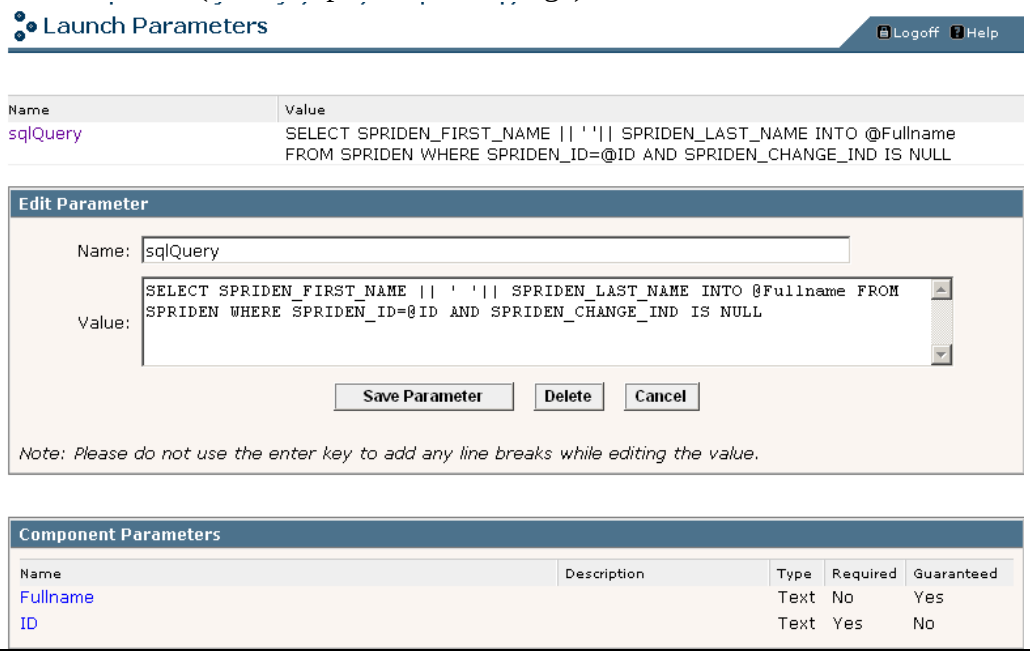
Jump to TOC

Solution

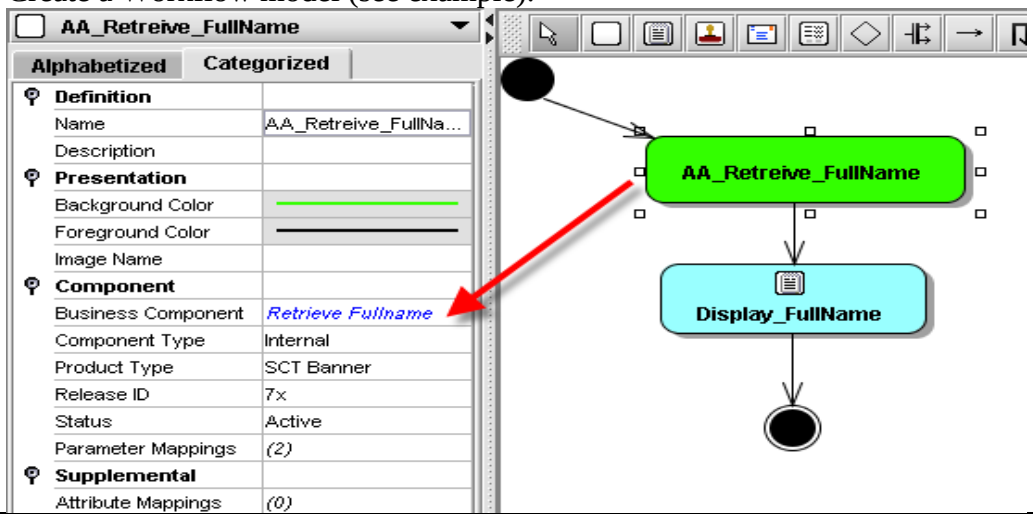
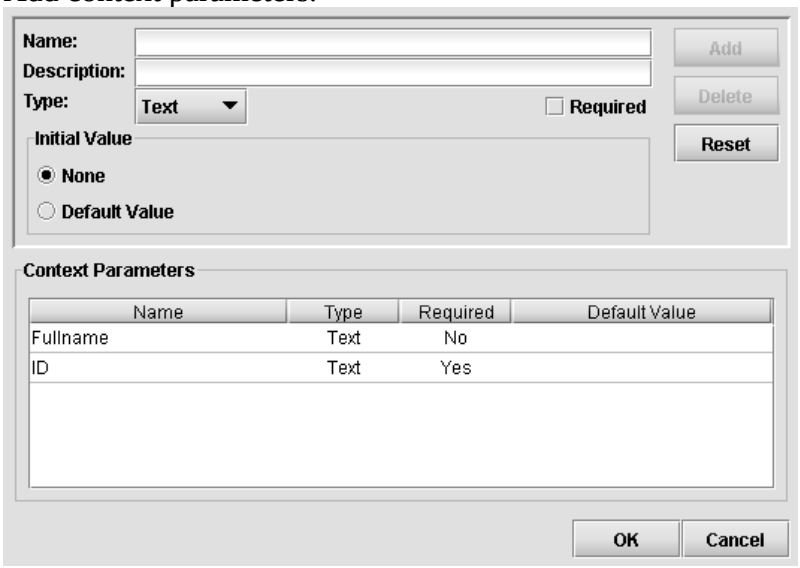
Step	Action
Create a sqlQuery business component	
1	Open Workflow and log in. Login: your_username Password: your_password
2	Click on the *Business Component Catalog* under the **Administration** tab.
3	Click on the **Add New Category** button.
4	Enter a category name in the **Category Name** field.
5	Enter a description in the **Description** field.
6	Click the **Save Category** button.
7	Click the **Add Component** button in the **Business Components** block.
8	Enter business component information.
9	Click the **Save Component** button.
10	Click the **Add Parameter** button in the **Parameters** block.
11	Enter applicable parameters.
12	Enter the name of the parameter in the **Name** field.
13	Click the **Add Parameter** button.
14	Click the **Back** button to return to the main component screen.
15	Click on **Client Launch Parameters.**

Solution, continued

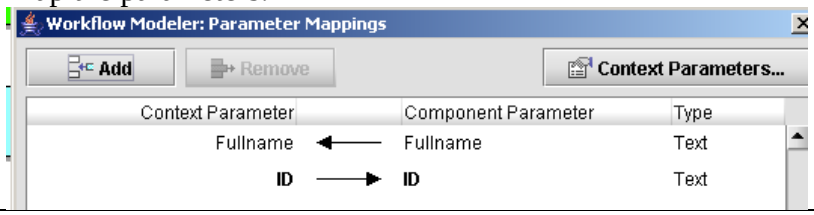
Step	Action
16	Enter "sqlQuery" in the **Name** field (this is case sensitive). Construct the SQL query in the **Value** field (see example in screen image).

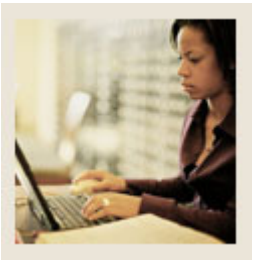


Solution, continued

Step	Action
Test the sqlQuery business component	
1	Create a Workflow model (see example). 
2	Attach the activity, *Retrieve_FullName*, to the SQL query business component.
3	Add context parameters.  Note: ID is required to start the model.

Solution, continued

Step	Action
4	Map the parameters. 
5	Create a manual activity called *DisplayFullName*. 



Section C: Defining Business Components

Lesson: Understanding SQL Procedures

Jump to TOC

Introduction

In this section, you will learn about the SQL procedures as well as understand how data types and null values are passed. In addition, you will gain a better understanding of grants needed for procedures as well as create a stored procedure (SQL procedure).

Modularizing code

In the previous section, you learned about SQL queries. Queries are great for simple tasks to return a piece of data. Since the queries are not stored in the database they must be recompiled each time they are run. For more complex programming logic, a stored procedure is a better bet. Stored procedures also allows for modularizing code. The importance of modularizing code is that it segments large, complex processes into smaller, simpler blocks of code.

Advantages	Disadvantages
• Reusable • Manageable • Readable • Reliable	• Higher level of sql skills required • Oracle access needed

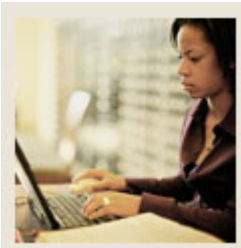
SQL Scripts

Autonomous blocks are NOT stored Oracle objects. They are PL/SQL blocks that have no name. They are compiled and evaluated every time a script is run. And, they are NOT accessible in Workflow.

Stored subprograms

Subprograms are stored within a database. They may be called from any application that allows the use of PL/SQL. There are three types of stored subprograms

- procedures
- functions
- packages.



Section C: Defining Business Components

Lesson: Understanding SQL Procedures (Continued)

Jump to TOC

SQL procedure syntax

SQL procedures perform specific actions. When writing syntax for SQL procedures, remember that:

- The keyword DECLARE is not used.
- The specification begins with CREATE OR REPLACE PROCEDURE and ends with the procedure name or parameter list.
- The body begins with IS and ends with END <(optional) procedure name>.
- Parameter declarations are optional.
- There are no constraints on parameter data types.

Syntax example:

```
CREATE [OR REPLACE]
PROCEDURE name [(parameter [,parameter])] IS
      [local declarations]
BEGIN executable statements
[EXCEPTION exception handlers]
END [name];
      (where <parameter> stands for:
            parameter_name [IN|OUT|IN OUT]
                  datatype [{:= | DEFAULT} expr])
```

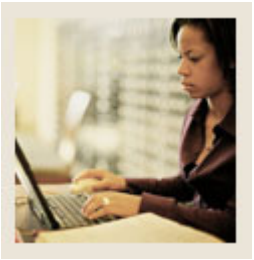
Storing SQL procedures

SQL procedures are stored as compiled code. They may be called from multiple applications such as other PL/SQL programs, Oracle forms, Pro*C, Pro*Cobol, and Workflow. Once created, a SQL procedure may be called from another PL/SQL block.

Note: It is important to decide which schema will be used to store SQL procedures for Workflow. WFOBJECTS is delivered schema for storing. For this training, objects will be stored in your WFUSER*xx* schema.

The following syntax is used when storing a procedure.

```
CREATE {OR REPLACE} PROCEDURE <schema.procedure_name> IS
…    -- body goes here
```



Section C: Defining Business Components

Lesson: Understanding SQL Procedures (Continued)

Jump to TOC

Passing parameters

There are several key points to remember when passing parameters to and from the database to Workflow.

- When parameters are passed into a SQL procedure, any constraints attached to it are also passed.
- It is illegal to constrain the CHAR or VARCHAR2 variable with length and numeric value with precision and scale.
- PL/SQL has no explicit limit of parameters.

Actual versus formal parameters

There are two types of parameters.

- **Actual** refers to the parameter list in the calling statement. These are the "actual" values.
- **Formal** refers to the parameter list in the procedure. These are just placeholders.

Parameter modes

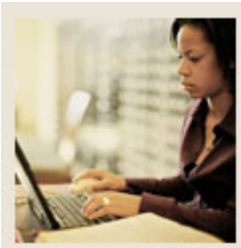
There are three types of parameter modes.

- IN – Value of parameter passed to subprogram (read-only). It is the default mode when not explicitly defined.
- OUT – Value of parameter being passed ignored (write-only). It is derived from within the subprogram and passed back. The content of the formal parameter is assigned to the actual parameter.
- IN OUT – Combination of IN and OUT. The value of the parameter may be passed in, reassigned a value within the subprogram, and passed back.

Handling compilation errors

If compilation error occurs, use `SQL> SHOW ERRORS`

Note that the stored procedure cannot be easily corrected directly when errors occur. You will need to re-run the file that caused the error after correcting the error(s). The REPLACE option allows the existing stored database procedure or function to be replaced.



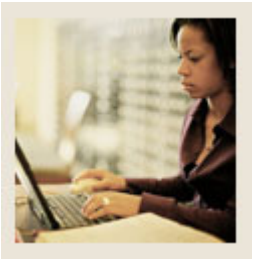
SQL procedure grants

Schema in which SQL procedure is compiled needs explicit select/update/insert permission. Grants cannot be applied via roles.

Example: grant select on gpvent1 to WFOBJECTS
grant execute on f_format_name to WFOBJECTS

WFAUTO must be granted execute permission for the procedure.

Example: grant execute on p_get_person_data to WFAUTO



Section C: Defining Business Components

Lesson: Exercise 10 – Create a SQL Procedure

Jump to TOC

Exercise 10 – Create a SQL procedure

1. Create a procedure in your WFUSERxx schema that returns indicators and address information for determining if a person is an enrolled student, financial aid applicant, employee, or constituent.

Additional information

2. Name your procedure P_get_person_data*xx*

3. Input parameter is PIDM and Aid Year

4. Output parameters are student_ind, employee_ind, finaid_app_ind, constituent_ind, street1, street2, city, state, zip.

5. Grant execute on P_get_person_data*xx* to WFAUTO

Section C: Defining Business Components

Lesson: Exercise 10 – Solution

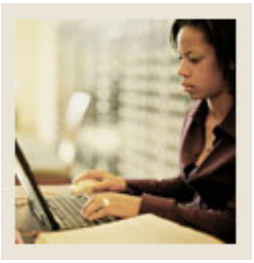
Jump to TOC

Solution

```
CREATE OR REPLACE PROCEDURE P_Get_Person_Data
(pidm IN NUMBER,aidyear IN VARCHAR2, StudentInd OUT VARCHAR2, EmployeeInd OUT
VARCHAR2,
FinAidInd OUT VARCHAR2, ConstituentInd OUT VARCHAR2,
Street1 OUT VARCHAR2,Street2 OUT VARCHAR2, City OUT VARCHAR2,
State OUT VARCHAR2,Zip OUT VARCHAR2) AS

BEGIN
SELECT F_Student_Enrollment_Ind(pidm) INTO StudentInd FROM dual;
SELECT F_Payroll_Employee_Ind(pidm) INTO EmployeeInd FROM dual;
SELECT F_Alumni_Constituent_Ind(pidm) INTO ConstituentInd FROM dual;
SELECT F_Finaid_Applicant_Ind(pidm,aidyear, '') INTO FinAidInd FROM dual;

SELECT spraddr_street_line1, spraddr_street_line2, spraddr_city,
spraddr_stat_code, spraddr_zip
         INTO street1, street2, city, state, zip FROM spraddr
         WHERE
         spraddr_pidm=pidm AND spraddr_atyp_code='MA' AND
         (spraddr_to_date IS NULL OR spraddr_to_date > SYSDATE) AND
         spraddr_status_ind IS NULL;
      END P_Get_Person_Data;
```

After the procedure is compiled, issue a grant to wfauto.

```
sql> grant execute on P_Get_Person_Data to wfauto;
```



Section C: Defining Business Components

Lesson: Exercise 11 – Create a SQL Procedure Business Component

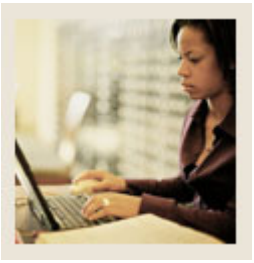
Jump to TOC

Exercise 11 – Create a business component for a SQL procedure

1. Create a business component for your P_get_person_data*xx* procedure.

2. Build a Workflow model to test your procedures with two activities:

   - The first activity links to your business component.
   - The second activity is a manual activity that displays the output returned.



Section C: Defining Business Components

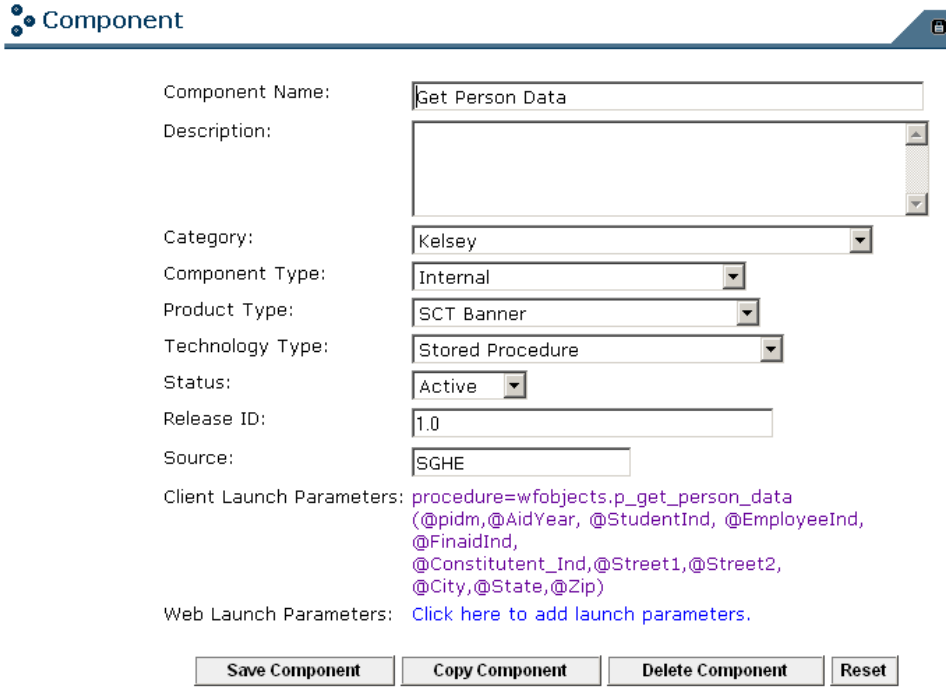
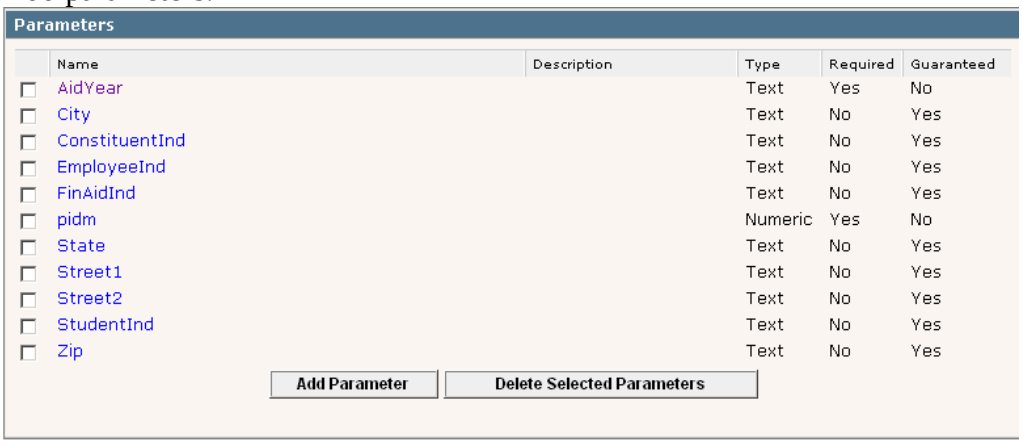
Lesson: Exercise 11 – Solution

Jump to TOC

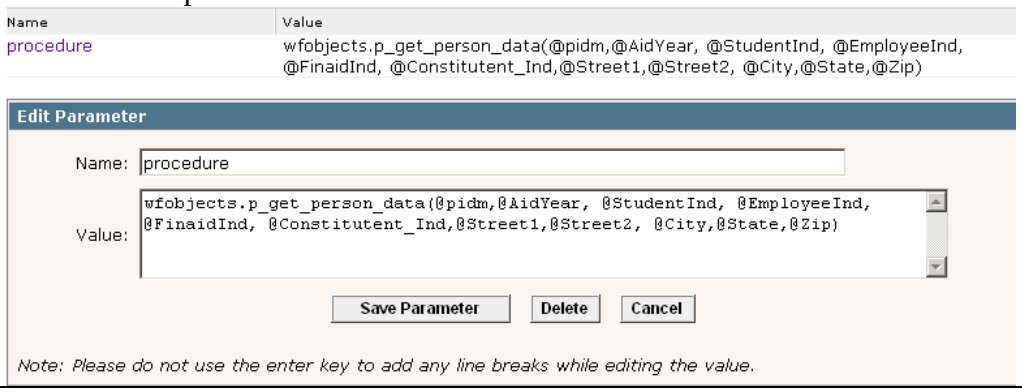
Solution

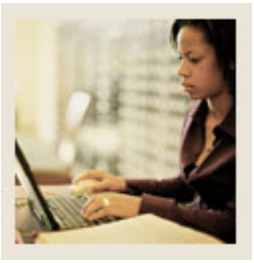
Step	Action
Create a business component for a SQL procedure	
1	Click on the *Business Component Catalog* under the **Administration** section. 
2	Click on the category for your user.

Solution, continued

Step	Action
3	Add a business component (see following example). 
4	Add parameters.  Note: PIDM is required because it is input to the procedure. The Indicator Parameters are guaranteed because the procedure is sending output back to the model – RINGO.

Solution, continued

Step	Action
5	Construct the procedure call. 
6	Create a model similar to the one in the SQL Query exercise to test your business component.



Section C: Defining Business Components

Lesson: Understanding Data Types and Dates

Jump to TOC

Purpose

In this lesson you will gain a better understanding of how data types are passed to and from Workflow business components as well as learn how dates are handled in Workflow.

Parameter data types

Data types of Actual and Formal parameters must match. All Workflow data types are passed in and out of Workflow as strings. Values passed to Workflow may NOT be null.

Data Type	Workflow Parameter Format	Oracle Format
Text	Passed as a string, changed to number	number
Text	Passed as a string, no formatting done	varchar2
Date	Passed as string	varchar2
Numeric	String representation of number (most systems will automatically convert to a numeric field by default).	varchar2 or number
Boolean	Passed as the string ‘true’ or ‘false’	varchar2

Numbers

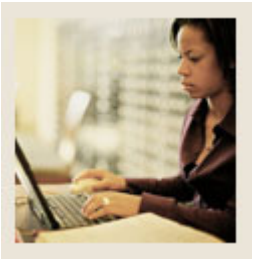
All numbers are passed as strings, but will automatically convert to numeric.

Boolean value

Boolean values must return either “true” or “false” to Workflow.

Syntax for inserting a Boolean value is as follows.

```
sqlQuery=select decode(ACTIVE, 'Y’, ‘true’, ‘N’,
‘false),'') into @active from USER where USER.ID = @id
```



Dates in Workflow

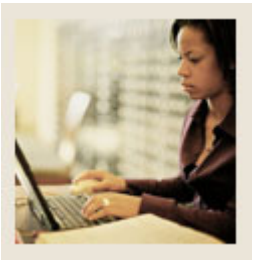
All dates are passed as strings. Be sure to use data type of varchar2 in SQL procedures.

Dates are formatted using a 24-hour clock method: dd-mon-yyyy hh24:mi:ss.

Example: 17-Aug-2006 15:23:00

Syntax for comparing an oracle date with a workflow date follows.

```
sqlQuery=
select to_char(spriden_activity_date,'DD-MON-RRRR HH24:MI:SS'),
     case  when spriden_activity_date>
          to_date(@compare_date,'DD-MON-RRRR HH24:MI:SS')  then
     'Y' else 'N'
     end
into @spriden_act_date,@Compare_ind from spriden where
spriden_pidm=@pidm and spriden_change_ind is null

Note that the oracle date must be converted to character
(spriden_activity_date), whereas the workflow date must be
converted to a date format (@compare_date).
```



Section C: Defining Business Components

Lesson: Creating a Function

Jump to TOC

Purpose

In this lesson, you will learn about the structure of functions and where to find delivered functions. You will also create a sqlQuery business component that calls a function and gain a better understanding of the privileges required to compile and execute a function.

Introduction

Functions are one of the three types of stored subprograms. They

- create libraries of customized calculations
- require fewer modifications of programs
- may be called within a sqlQuery SELECT statement

Example: SELECT f_format_name(1234,'FML') FROM dual;

Syntax for functions

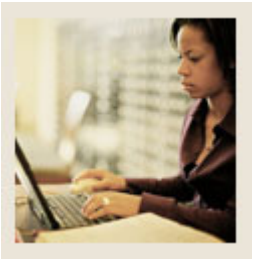
Functions are similar to procedures in structure. They compute and return one value. Syntax for functions is as follows.

Example:

```
FUNCTION name [(parameter [, parameter, ...])]
     RETURN DATATYPE IS [local declarations]
     BEGIN executable statements
          [EXCEPTION exception handlers]
     END [name];
```

Note: <parameter> stands for

```
     parameter_name [IN|OUT|IN OUT]
       datatype [{:= | DEFAULT} expr]
```



Multiple return statements

Function will stop when it reaches a RETURN statement. A function can contain multiple return statements.

Example:

```
IF … THEN
   RETURN 'TRUE';
ELSE
   RETURN 'FALSE';
END IF;
```

Function privileges

Table select privileges must be granted to the schema owner.

Example: `sql> grant select on spriden to wfobjects`

Execute privileges must be granted to WFAUTO.

Example: `sql> grant execute on f_new_function to wfauto;`

Subprogram dependencies

Changing dependencies after compilation (i.e., altering or dropping a table) could make the subprogram invalid. You can query against the data dictionary view ALL_OBJECTS to verify validity.

Example:

```
SQL> SELECT OBJECT_NAME, STATUS
    FROM ALL_OBJECTS
   WHERE OBJECT_NAME = '&OBJECT_NAME';
```

Section C: Defining Business Components

Lesson: Creating a Function (Continued)

Jump to TOC

Workflow and functions

Functions can be called from procedures OR accessed directly from a sqlQuery business component.

Example: Judith M Maxwell

```
sqlQuery=SELECT F_FORMAT_NAME(@pidm,'FML') into @fullname FROM
dual
```

Example: Maxwell, Judith

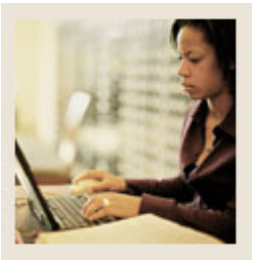
```
sqlQuery=SELECT F_FORMAT_NAME(@pidm,‘LF30') into @fullname FROM
dual
```

Banner delivered functions

Banner delivered functions are owned by BANINST1. The names begin with “F_”. Any delivered functions can be used in Workflow. There are 550-plus delivered functions so don’t reinvent the wheel.

Example:

```
SELECT   * FROM all_objects
          WHERE
  object_type ='FUNCTION' AND
  owner ='BANINST1‘;
```



Section C: Defining Business Components

Lesson: Exercise 12 – Create a Function

Jump to TOC

Exercise 12 – Create a function

1. Write a function to return an indicator of 'true' if all addresses for an address type are inactive.

   - Input is PIDM and Address Type.
   - Output is true if all addresses are inactive, or false if there is an active address.

2. Create a sqlQuery business component to call your function.

3. Test your component by building a model with two activities.

   - A component activity linking to your sqlQuery business component.
   - A manual activity to display results.

Section C: Defining Business Components

Lesson: Exercise 12 – Solution

Jump to TOC

Solution

```
CREATE OR REPLACE FUNCTION F_INACTIVE_ADDR_CHECKxx(p_pidm NUMBER, p_addrtype
VARCHAR2 )
RETURN VARCHAR2
IS
  v_inactive_addr_ind VARCHAR2(5) :='true';
  v_pidm NUMBER;

 /* Look for inactive addresses for an address type and pidm */
             CURSOR c_address_inactive IS
             SELECT DISTINCT spraddr_pidm FROM spraddr a WHERE
             spraddr_pidm=p_pidm
             AND SPRADDR_ATYP_CODE=p_addrtype
             AND NOT EXISTS
                   (SELECT 'x' FROM spraddr b WHERE
                   a.spraddr_pidm=b.spraddr_pidm
                        AND b.spraddr_status_ind IS NULL
                        AND a.spraddr_atyp_code=b.spraddr_atyp_code);
BEGIN
    OPEN c_address_inactive;
    FETCH c_address_inactive INTO v_pidm;
    CLOSE c_address_inactive;
    IF v_pidm IS NULL THEN
         v_inactive_addr_ind :='false';
    END IF;
    RETURN v_inactive_addr_ind;
      END;
```

sql> grant execute on `F_INACTIVE_ADDR_CHECKxx to wfauto;`

Section D: Testing and Debugging

Lesson: Overview

Jump to TOC

Purpose

In this section, you will learn how to test and debug your procedures and functions.

Objectives

On completion of this session, you should be able to

- state the value of debugging procedures before they are used in a business component
- create a simple script to call procedures and display output
- describe the Oracle package DBMS_OUTPUT.

Section contents

Section D: Testing and Debugging

Lesson: Testing Objects

Jump to TOC

Introduction

By testing Oracle objects using a SQL wrapper, you can identify errors before the procedure is called from a business component. Testing involves a three-step process:

1. Define parameters.
2. Call the procedure or function.
3. Display the output using dbms_output.

The syntax for the SQL wrapper is as follows.

```
SET SERVEROUTPUT ON
DECLARE
        email_address varchar2(80);
        pidm number :=23456' ;
BEGIN
        DBMS_OUTPUT.ENABLE(200000);
        p_get_email_address(pidm,email_address);
        DBMS_OUTPUT.PUT_LINE('Email Address for pidm ',pidm, 'is ',email_address);
END;
```

Screen output

The Oracle package DBMS_OUTPUT is enabled by the statement SET SERVEROUTPUT ON.

Output command options are listed in the table.

Output Command Options	
ENABLE	NEW_LINE
DISABLE	GET_LINE
PUT_LINE	GET_LINES
PUT	

Note: Line output must be in parentheses and fields must be concatenated.

Troubleshooting

If you run a model and get the following error, it means that the engine cannot find the procedure.

Error: `"procedure_name is not declared"`

Check: the following:

- ❑ Is grant execute issued to WFAUTO?
- ❑ Is schema name included in the business component definition?

Section D: Testing and Debugging

Lesson: Exercise 13 – Test a SQL Procedure

Jump to TOC

Exercise 13 – Test a SQL procedure

1. Create a script to call your P_get_person_dataxx procedure and display the output using the DBMS_OUTPUT package.

2. Run your script.

Note: You may also be able to test your procedure from a TOAD or sqlDeveloper GUI.

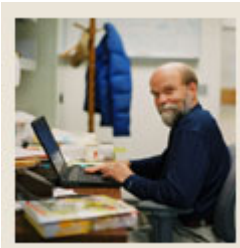
Section D: Testing and Debugging

Lesson: Exercise 13 – Solution

Jump to TOC

Solution

```
SET SERVEROUTPUT ON
DECLARE
      student_ind varchar2(1);
      employee_ind varchar2(1);
      finaid_app_ind varchar2(1);
      const_ind varchar2(1);
      street1 varchar2(30);
      street2 varchar2(30);
      city varchar2(20);
      state varchar2(3);
      zip varchar2(10);

      pidm number :=35 ;
      aidyear := '0607';

      BEGIN
      DBMS_OUTPUT.ENABLE(200000);
      P_get_person_dataxx
      (pidm, aidyear, student_ind, employee_ind, finaid_app_ind,
      const_ind, street1, street2, city, state, zip)
DBMS_OUTPUT.PUT_LINE('pidm= '||pidm||' Student Indicator = ' ||
student_ind ||'Employee Indicator = '||employee_ind||' 'Financial
Aid Indicator = ' ||finaid_app_ind||' Constituent Indicator = '
|| const_ind);
DBMS_OUTPUT.PUT_LINE('Address is ' ||street1 ||', '||street2||', '
||city|| ', ''||state|| ', ' ||zip;
END;
```



Section E: Error Handling

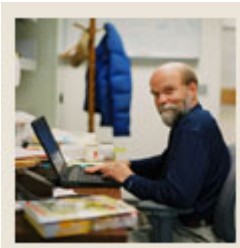
Lesson: Overview

Jump to TOC

Purpose

In this section, you will learn about the advantages of exception handling as well as identify types of exception handling and PL/SQL error information functions. You will also learn how to pass error messages to Workflow.

Section contents



Section E: Error Handling

Lesson: Understanding Exceptions

Jump to TOC

What is exception processing?

Exception processing is used to set default values if no records are returned.

The advantages of exception handling include

- event-driven handling of errors
- separation of error-processing code
- improved reliability of error handling.

Type of exceptions

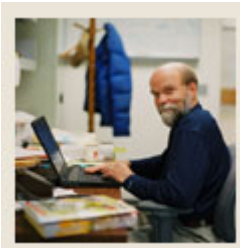
There are three types of exceptions

- named system exceptions
- named programmer-defined exceptions
- unnamed system exceptions.

A **named system exception** is automatically raised by Oracle. Several standard named exceptions are shown in the table.

EXCEPTION NAME	ORACLE ERROR	SQLCODE Value
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
STORAGE_ERROR	ORA-06500	-6500
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Note: See www.ora-code.com for a complete list of Oracle Error Codes and descriptions.



Section E: Error Handling

Lesson: Understanding Exceptions (Continued)

Jump to TOC

Type of exceptions, continued

The syntax for a **named system exception** follows.

```
WHEN <exception_name> [OR <exception name> …]
      THEN <sequence of statements>
     …
      [WHEN OTHERS THEN  --if used, must be the last handler
     <sequence of statements>]
```

Example:

```
WHEN NO_DATA_FOUND THEN
     fullname := "No record found for this ID";
```

A **named programmer-defined exception** is declared and raised by a programmer. . Once it is raised manually, it is treated as a pre-defined internal exception. It is scoped just like variables in that exceptions declared in a block are local to a block.

The syntax for a named programmer-defined exception follows.

```
DECLARE my_exception  Exception
          . . .
          RAISE my_exception
```

Example: Tuition Waiver Dependent Check

```
DECLARE exc_too_old  EXCEPTION;
BEGIN
     IF f_calculate_age(sysdate,birthdate,'')>25
THEN RAISE exc_too_old;
END IF;
EXCEPTION
WHEN exc_too_old then
…….
END;
```

Section E: Error Handling

Lesson: Understanding Exceptions (Continued)

Jump to TOC

Type of exceptions, continued

An **unnamed system exception** associates a name to an Oracle error number. It is used to trap errors that are not pre-defined.

The syntax for an **unnamed system exception** follows.

```
PRAGMA EXCEPTION_INIT <your_exception_name,
Oracle error number>
```

Example:

```
DECLARE
     deadlock_detected EXCEPTION;
     PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
   BEGIN
     ... -- Some operation that causes an ORA-00060 error
     EXCEPTION
     WHEN deadlock_detected
     THEN -- handle the error
     END;
```

SQL code and SQLERRM

SQLCODE andSQLERRM

SQLCODE returns the Oracle error number of the exception. If it was a user-defined exception, a "1" is returned. SQLERRM returns the Oracle error message associated with the current SQLCODE value.

Note: Oracle error numbers may be used as argument.

SQLCODE and SQLERRM cannot be used within a SQL statement.

Handling null values

Null values CANNOT be used in Workflow guard conditions. You can assign a null value in the exception blocks.

Section E: Error Handling

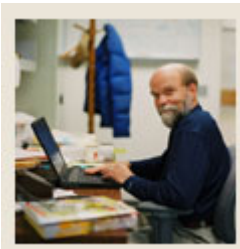
Lesson: Exercise 14 – Add an Exception Block

Jump to TOC

Exercise 14 – Add an exception block

1. Add an exception block to your procedure P_get_person_dataxx to check if no data is returned.

2. If NO_DATA_FOUND is true, assign the value ‘Address not on file’.to street line1.



Section E: Error Handling

Lesson: Exercise 14 – Solution

Jump to TOC

Solution

```
CREATE OR REPLACE PROCEDURE wfobjects.P_Get_Person_Data
(pidm IN NUMBER,aidyear IN VARCHAR2, StudentInd OUT VARCHAR2, EmployeeInd OUT
VARCHAR2,
FinAidInd OUT VARCHAR2, ConstituentInd OUT VARCHAR2,
Street1 OUT VARCHAR2,Street2 OUT VARCHAR2, City OUT VARCHAR2,
State OUT VARCHAR2,Zip OUT VARCHAR2) AS

BEGIN
SELECT F_Student_Enrollment_Ind(pidm) INTO StudentInd FROM dual;
SELECT F_Payroll_Employee_Ind(pidm) INTO EmployeeInd FROM dual;
SELECT F_Alumni_Constituent_Ind(pidm) INTO ConstituentInd FROM dual;
SELECT F_Finaid_Applicant_Ind(pidm,aidyear, '') INTO FinAidInd FROM dual;

SELECT spraddr_street_line1, spraddr_street_line2, spraddr_city,
spraddr_stat_code, spraddr_zip
         INTO street1, street2, city, state, zip FROM spraddr
         WHERE
         spraddr_pidm=pidm AND spraddr_atyp_code='MA' AND
         spraddr_to_date < SYSDATE AND
         spraddr_status_ind IS NULL;
      EXCEPTION
      WHEN NO_DATA_FOUND THEN
      street1 :='Address  not on file';
      END P_Get_Person_Data;
```

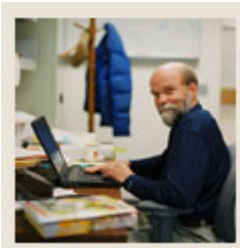
Section E: Error Handling

Lesson: Exercise 15 – Determine Table Constraints

Jump to TOC

Exercise 15 – Determine table constraints

1. Determine the foreign table constraints for the SCBCRSE table by querying the Oracle View all_constraints.



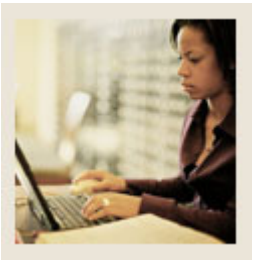
Section E: Error Handling

Lesson: Exercise 15 – Solution

Jump to TOC

Solution

```
Select * from all_constraints where table_name='SCBSCRSE';
```



Section F: Creating Database Packages

Lesson: Overview

Jump to TOC

Introduction

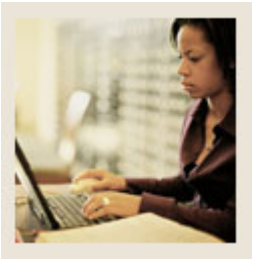
In this section, you will learn how to bundle subprograms into packages.

Objectives

On completion of this session, you should be able to

- create database packages
- list the security benefits that packages offer
- list some built-in packages that Oracle provides.

Section contents



Section F: Creating Database Packages

Lesson: About Packages

Jump to TOC

Introduction

As you create stored functions and procedures, you will be able to reduce the amount of code that is in each application. However, when you begin to rely heavily upon stored subprograms, you will soon realize that it will be difficult to know what each subprogram is used for. Of course, you will want to add comments in each subprogram. But beyond this, what can you do? You can bundle subprograms into packages. Not only will packages allow us to bundle subprograms, but also global cursors and variables.

Benefits of packages

- **Packages promote the object-oriented model**

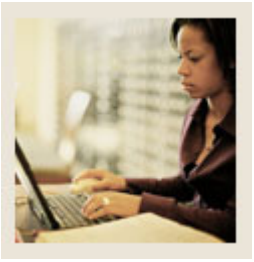
Although packages do not support every concept in the object-oriented design model, they do provide for some principles such as object hierarchy. The Oracle RDBMS automatically tracks the validity of all program objects (procedures, functions, and packages) stored in the database. It determines what other objects that program is dependent on, such as other packages. If a dependent object changes, then all programs that rely on that object are flagged invalid. Then, the dependent objects are automatically re-compiled until all dependencies are met.

- **Performance benefits**

When one object in a package is referenced for the first time, the entire package (already compiled and validated) is loaded into the Shared Global Area (SGA) of the database. All other package elements are thereby made immediately available for future calls to the package. PL/SQL does not have to keep retrieving program elements or data from disk each time a new object is referenced. In a distributed environment where packages are executed across a local area network, minimization of network traffic can boost performance quite substantially.

- **Package security**

As a PL/SQL object, security on the package is limited to the same principles as the other objects; grant privileges, object owners, etc. However, by nature, the package can secure the objects it contains thereby hiding some of the most detailed aspects of your programs. When you build a package, you decide which of the elements are public (referenced outside the package) and which are private (available only within the package itself) using the package specification and the package body. Public objects are defined within the package specification. Private objects are defined within the package body. So, by defining an object as private, you can essentially protect the most confidential of business rules.



Section F: Creating Database Packages

Lesson: About Packages (Continued)

Jump to TOC

Package structure

The package consists of two distinct structures:

- Specification – Just as in defining procedures and functions, the specification (or header) defines to other objects how this object will be referenced.

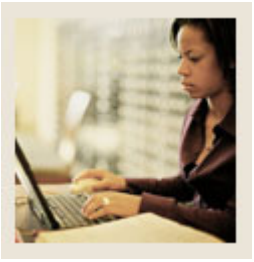
- Body – The body contains all of the code that executes the object. However, in functions and procedures, the keyword IS connects these two pieces. For a package, the specification and the body are not connected - they are separate, distinct code structures.

Package specification

To bundle PL/SQL objects together, the first thing to do is declare which pieces of your package are available to other applications. In other words, which objects are public, what type of objects they are, and what are the parameters expected whenever a programmer decides to use your package.

Syntax for a package specification follows.

```
CREATE OR REPLACE PACKAGE package_name
IS
[ declarations of public variables and types ]
[ specifications of public cursors ]
[ specifications of modules (i.e. functions and procedures) ]
END [ package_name ];
```



Package body

Once the specification declares to the database what to expect from your package, you can decide whether you need to code a package body. Does your specification declare any cursors? Does your specification declare any functions or procedures? If so, then there is PL/SQL code which has to be assigned to those objects before they are complete.

The role of the package body is to contain the code behind those objects that require the specific PL/SQL language constructs.

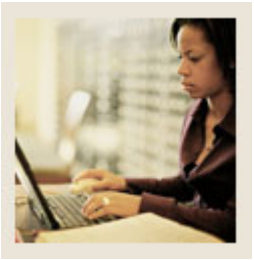
Syntax for a package body follows.

```
CREATE OR REPLACE PACKAGE BODY package_name
     IS
     [ declarations of public variables and types ]
     [ specifications of public cursors ]
     [ specifications of modules (i.e. functions and
procedures)]

     [ BEGIN
     executable statements ]
     [ EXCEPTION
     exception statements]
     END [ package_name ];
```

Synchronize the specification and the body

It is imperative that you keep the package specification synchronized with the body and vice versa. If you do not, the compiler will generate the following error:

> *PLS-00232: subprogram 'name' is declared in a package specification and must be defined in the package body*



Synchronize the specification and the body, continued

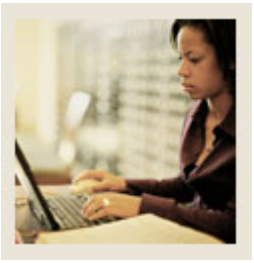
Consider this, however; if you only had variables, constants, and exception types declared in a package specification, would the package body be required? The answer is no, because the specification has declared to the database that there are no 'incomplete' objects within the package.

Variables, constants, and exceptions are complete once they are defined within the declaration section of any PL/SQL block. By building such a package specification you can make a number of standard variables, constants, and/or exceptions available for use throughout your applications simply by declaring them once within a package.

Overloading packages

A package can be overloaded, which means that more than one procedure or function has the same name, but with different parameters. Based on the datatype or number of parameters, Oracle will be able to deduce which subprogram needs to run.

In Workflow, each procedure or function must be associated with ONE business component. If two procedures have the same name, each procedure must have a business component associated with it. This is because the parameter types or the number of parameters may be different.



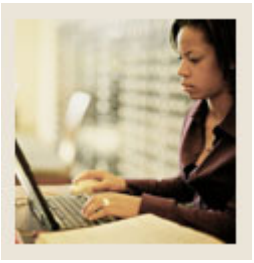
Section F: Creating Database Packages

Lesson: Exercise 16 – Create a Package

Jump to TOC

Exercise 16 – Create a package

1. Create a package that includes the procedure and function you wrote in this training.
2. Grant execute access to WFAUTO.



Section F: Creating Database Packages

Lesson: Exercise 16 – Solution

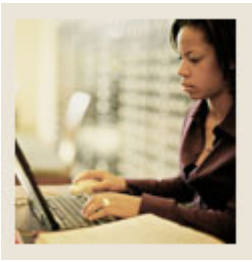
Jump to TOC

Solution

```
CREATE OR REPLACE PACKAGE WFK_ADDRESS_CHANGExx
IS
         FUNCTION F_Inactive_Addr_Check(p_pidm NUMBER, p_addrtype VARCHAR2 )
RETURN VARCHAR2;
         PROCEDURE P_STATE_CHANGExx(full_name IN VARCHAR2, id IN VARCHAR2);
         PROCEDURE P_GET_PERSON_INDICATORSxx
         (p_pidm IN NUMBER, p_student_ind OUT VARCHAR2, p_employee_ind OUT
VARCHAR2, p_finaid_app_ind OUT VARCHAR2, p_const_ind OUT VARCHAR2);
END WFK_ADDRESS_CHANGExx;
/
CREATE OR REPLACE PACKAGE BODY WFK_ADDRESS_CHANGExx
IS
FUNCTION F_Inactive_Addr_Check(p_pidm NUMBER, p_addrtype VARCHAR2 )
RETURN VARCHAR2
IS
  v_inactive_addr_ind VARCHAR2(5) :='true';
  v_pidm NUMBER;

 /* Look for inactive addresses for an address type and pidm */
 CURSOR c_address_inactive IS
             SELECT DISTINCT spraddr_pidm FROM spraddr a WHERE
             spraddr_pidm=p_pidm
             AND SPRADDR_ATYP_CODE=p_addrtype
             AND NOT EXISTS
(SELECT 'x' FROM spraddr b WHERE a.spraddr_pidm=b.spraddr_pidm
                        AND b.spraddr_status_ind IS NULL
                        AND a.spraddr_atyp_code=b.spraddr_atyp_code);
BEGIN
    OPEN c_address_inactive;
    FETCH c_address_inactive INTO v_pidm;
    CLOSE c_address_inactive;
    IF v_pidm IS NULL THEN
         v_inactive_addr_ind :='false';
    END IF;
    RETURN v_inactive_addr_ind;
END;
PROCEDURE P_FA_STATE_CHANGExx(full_name IN VARCHAR2, id IN VARCHAR2) IS

  v_Params                    Gokparm.t_parameterlist;
  event_code                  gtveqnm.gtveqnm_code%TYPE;
```

Solution, continued

```
BEGIN

  IF  Goksyst.f_isSystemLinkEnabled('WORKFLOW') THEN
     event_code
:=SUBSTR(Gokevnt.F_CheckEvent('WORKFLOW','FA_STATE_CHANGE'),1,20);

      --  pass parameters to the event
      v_Params(1).param_value := 'FA_STATE_CHANGE';
      v_Params(2).param_value := '';
      v_Params(3).param_value := 'FA State Change for '|| full_name ||'
'||id;
      v_Params(4).param_value := id;
      v_params(5).param_value := full_name;
      Gokparm.Send_Param_List(event_code, v_Params);

  END IF;
END;

PROCEDURE P_Get_Person_Dataxx
(pidm IN NUMBER,aidyear IN VARCHAR2, StudentInd OUT VARCHAR2, EmployeeInd OUT
VARCHAR2,
FinAidInd OUT VARCHAR2, ConstituentInd OUT VARCHAR2,
Street1 OUT VARCHAR2,Street2 OUT VARCHAR2, City OUT VARCHAR2,
State OUT VARCHAR2,Zip OUT VARCHAR2) AS

BEGIN
SELECT F_Student_Enrollment_Ind(pidm) INTO StudentInd FROM dual;
SELECT F_Payroll_Employee_Ind(pidm) INTO EmployeeInd FROM dual;
SELECT F_Alumni_Constituent_Ind(pidm) INTO ConstituentInd FROM dual;
SELECT F_Finaid_Applicant_Ind(pidm,aidyear, '') INTO FinAidInd FROM dual;

SELECT spraddr_street_line1, spraddr_street_line2, spraddr_city,
spraddr_stat_code, spraddr_zip
         INTO street1, street2, city, state, zip FROM spraddr
         WHERE
         spraddr_pidm=pidm AND spraddr_atyp_code='MA' AND
         spraddr_to_date < SYSDATE AND
         spraddr_status_ind IS NULL;
END P_Get_Person_Data;
END WFK_ADDRESS_CHANGExx;
```



Section G: Using Banner Delivered API's

Lesson: Overview

Jump to TOC

Introduction

In this section, you will learn how use the Banner delivered API's for querying and updating tables.

Objectives

On completion of this session, you should be able to

- locate API packages
- query or update Banner tables using the API's..

Section contents

Section G: Using Banner Delivered API's

Lesson: About Banner API's

Banner APIs

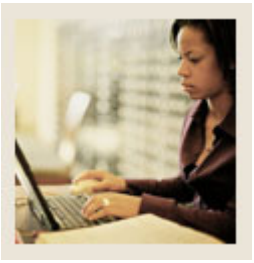
The Baninst1 Schema is the store house of all the functions, procedures and packages for all the Banner modules. Included are specific packages for Application Programming Interface (API). The purpose of these API's is to enable any third party application to update Banner tables and follow the same processing rules the interactive Banner forms use. If updating tables from Workflow, a recommended best practice is to use the API's if available.

All the API's follow the same format. Listed here are the standard functions and procedures used in each package.

```
Functions:
    F_API_VERSION
      Return the current version number of the business
      entity API (not the file version number).
    F_EXISTS
      Determine if a record in the table exists.
    F_ISEQUAL
      Tests of two records of the same type are equal.
    F_QUERY_ALL
      Returns a REF CURSOR for a set of records
    F_QUERY_BY_ROWID
      Returns a REF CURSOR for one record
    F_QUERY_ONE
      Returns a REF CURSOR for one record
    F_QUERY_ONE_LOCK
      Returns a REF CURSOR for one record and also locks the record

  Procedures:
    P_CREATE
      Inserts a record into the table.
    P_DELETE
      Deletes a record from the table.
    P_LOCK
     Locks the current record for update.
    P_UPDATE
      Updates a record in the table.

In order to do a table update, the original record must be sent in as a
parameter to the P_UPDATE procedure. Here is an example for retreving the a
current email record.
```

```
select goremal_email_address into old_address from goremal
where goremal_pidm = pidm and goremal_emal_code = email_type
        and goremal_status_ind = 'A';
```

Once the original record is retrieved, a call can be made to P_UPDATE to update the record.



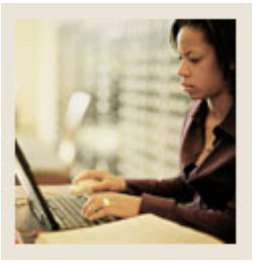
Section G: Using Banner Delivered API's

Lesson: Exercise 17 – Update email address using API's

Jump to TOC

Exercise 17–

- **Write a procedure to call the Banner API's to update and email address.**
  - Retrieve the original record
  - Inactive the record
  - Create a new email record with the new address



Section G: Using Banner Delivered API's

Lesson: Exercise 17 – Update email address using API's

Jump to TOC

Exercise 17– Solution

```
create or replace procedure p_email_updatexx(
        pidm IN spriden.spriden_pidm%TYPE,
        email_type IN goremal.goremal_emal_code%TYPE,
        new_email_address IN goremal.goremal_email_address%TYPE,
        err_msg OUT varchar2
) is

old_address goremal.goremal_email_address%TYPE;
email_update varchar2(1) := 'Y';
rowid_out gb_common.internal_record_id_type;

begin
        begin
                select goremal_email_address into old_address from goremal
                where goremal_pidm = pidm and goremal_emal_code = email_type
                        and goremal_status_ind = 'A';
                exception
                        when no_data_found then
                        email_update := 'N';
        end;

        begin
                if email_update = 'Y' then
                gb_email.p_update(pidm,email_type,old_address,'I','N','wfusrexx','','Y','test','');
                end if;
        end;

        begin
                gb_email.p_create(pidm,email_type,new_email_address,'A','Y','wfuserxx','added',
                'Y','test add',rowid_out);
                exception
                when others then
                err_msg:= substr(sqlerrm,1,200);

        end;
end p_email_updatexx;
```

Section G: Workflow Migration

Lesson: Overview

Jump to TOC

Introduction

In this section, you will learn how to migrate Workflow models from a development instance to a production instance.

Objectives

On completion of this session, you should be able to

- state how to export triggers and procedures to another instance
- discuss how to determine grants that need to be reapplied in the production instance
- describe how to export Banner event definitions
- identify how to export/import the Workflow models.

Section contents

Section G: Workflow Migration

Lesson: Migration Activities

Jump to TOC

Migration activities

Migrating Workflow models from a development instance to a production instance requires a number of steps, some from the Workflow Server, some from Oracle, and some from within Workflow.

Summary of Steps

Where Executed	Task	Purpose	Commands used
1. WF devl Server	Activate the model(s). Export all WF	Export everything from the development instance of WF including models, roles, users, business components.	export
	Unzip file and Extract specific model	Extract only the model definition desired to migrate.	unzip, extracwd
	Copy xml file to WF Server Production		
2. WF devl Banner instance	Run parameter_load_script.sql	Create a script to port the Banner event definition files.	@ parameter_load_script.sql
	Copy or ftp files for creating triggers and procedures to prod Banner instance.	Port the sql scipts to the production environment.	
3. WF prod Banner instance	Execute sql files for creating triggers and procedures in prod Banner instance.	Recreate all oracle objects in the production instance.	
4. Production Workflow Web Login	Create Workflow Business Event		Event Wizard
	Create Workflow Business Process	Create Business Process, associate model with Business Event, add users, role associations and proxies	Add to Workflow Manually.

Section G: Workflow Migration

Lesson: Migration Activities (Continued)

Jump to TOC

Migrating the Banner Event

In Banner test environment:

1. Run parameter_load_script.sql to create Banner event set-up script. You will be asked to supply the following. This will create the event_*Banner_event_name*.sql file.
    a. &&EVENT_NAME = Banner event name
    b. &&GROUP_NAME = Banner parameter group name

2. Copy the event sql file to Banner production.

In Banner production environment:

1. Run Event Queue load procedure, event_*Banner_event_name*.sql. This will update the Banner Event tables and create the event in Banner.

2. Verify entries on Event Queue forms.

Migrating Triggers and Procedures

1. Apply and enable database triggers.
2. Move new procedures to production and compile.
3. Change Active indicator on GOREQNM when ready to start events and workflows.

Note: Do not activate events until all parts of new event and workflow have been moved to production.

Section G: Workflow Migration

Lesson: Migration Activities (Continued)

Jump to TOC

Migrating the Workflow

Note: See *Workflow Technical Integration Guide* for export, extractwd and import command instructions.

In Workflow test environment:

1. Activate the workflow.
2. Export workflow from test environment (full data export):
   a. Log on to the non-production workflow server
   b. Go to the workflow_home directory for the correct environment (i.e., test, pprd, prod),
   c. Go to the **.bin subdirectory** and execute the following commands.
   d. Create the export file- The **export** command will do a full data export of all the workflow data into a .zip file. Save the export file as a backup.
   e. **Unzip this file** (for UNIX users, the unzip utility is located in the /users/bin directory.

   Example: export wfroot (password) fullexp20060101.zip

3. Extract the desired workflow model using extractwd command.
4. Create the extract file. The **extractwd** command creates an extract of each model individually including associated roles and components.

   Example: extractwd [-withoutDependencies] -source <sourcefile> -target <targetfile> -processdef {<organization> <name> <version>}*

Note:
–withoutDependencies –Means do NOT include roles and components associated with the model. The default is to *include* roles and components**.**
sourcefile (i.e. fullexp20060101.xml) – Output from the export. Note: If you created the export file as a zip file, you will need to extract the .xml file first.
targetfile – New file name for individual model .xml file.
processdef – This is the name of the model. It must be exactly the same as in Workflow.
organization (e.g. Root) – It must be exactly the same as in Workflow.
version – Number

Example: extractwd fullexp20060101.xml myWorkflowModel.xml ModelName Root 0

Section G: Workflow Migration

Lesson: Migration Activities (Continued)

Jump to TOC

Migrating the Workflow, continued

Note: If your workflow model name includes spaces, use the –shell option to prompt for each.

Example:

extractwd -shell
<Target> myWorkflowModel.xml
<Source> fullexport20060101.xml
Organization> Root
Workflow Definition> ModelName
<Version> 0
<With Dependencies> Y

5. Copy workflow model xml file to Workflow production environment.

In Workflow production environment:

1. Import workflow model into production environment.
   a. Go to the workflow_home directory for the correct environment (i.e. Prod).
   b. Go to the **.bin** directory of the target WF environment.
   c. Run the **import** using the output .xml file from the extractwd command.

   Example: import wfroot (password) myWorkflowModel.xml

2. Review import error logs to ensure all objects you expected to move were imported correctly (roles, components). The log file will probably contain errors for objects that already existed in the new environment. Existing same-named objects are not overlaid.
   a. Manually apply necessary updates to any existing objects that were not imported. For example, changes in components may need to be applied manually.

3. Change workflow model status to Active.

Updating Workflow Event and Business Process

1. Use the Banner Event Wizard to build business event. See instructions above in this document.
    a. Add workflow association to business event.

2. Build business process in Enterprise Management.
    a. Add workflow association to business process.
    b. Add event association to business process.

Other Setup

If necessary, add new workflow users, add roles assignments to users, and add proxy assignments. Notify appropriate offices of new workflow and required responses when they receive workflow activities.

Section G: Workflow Migration

Lesson: Exercise 18 – Identify Migration Steps

Jump to TOC

Exercise 18 – Identify migration steps

1. To get ready for migration, list the steps to take on the Workflow server you are migrating from.

2. What are the steps to take in production?

Section G: Workflow Migration

Lesson: Exercise 18 – Solution

Jump to TOC

Solution

To get ready for migration, what are the steps to take on the workflow server you are migrating FROM.

1. Export the workflow models.
2. Unzip and extract specific model(s) to be migrated.
3. Run parameter_load_script.sql to create an event load script.
4. Copy the .xml files to the production server.
5. Copy all sql triggers and procedures to the production server.
6. Copy the event script to the production server.

What are the steps to take in production?

1. Import the xml model files.
2. Compile the triggers, procedures, and event load script.
3. Create the Workflow Business Event and Business Process.

Section H: Workflow Tables

Lesson: Understanding Table Types

Jump to TOC

Introduction

In the Workflow schema, there are three groups of tables

- definition and configuration tables
- engine tables
- archive tables

The **definition and configuration tables** include such items as process models, activities, and components.

The **engine tables** include information on completed and running Workflow instances. These tables start with eng

The **archive tables** are a repository for report generation

Section contents

Section H: Workflow Tables

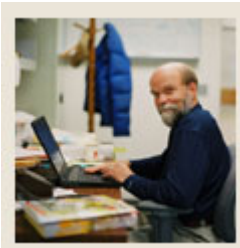
Lesson: Exercise 19 – Reporting

Jump to TOC

Exercise 19 – Reporting

1. Create a sqlQuery to list all users with no email address.

2. Create a sqlQuery to list **running processes and the date started.**



Section H: Workflow Tables

Lesson: Exercise 19 – Solution

Jump to TOC

Solution

1.

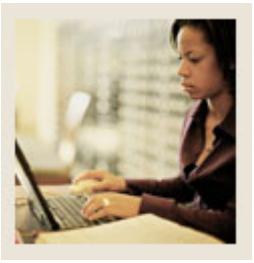
```
SELECT logon,last_name,first_name  FROM workflow.wfuser WHERE email_address
IS NULL;
```

2.

Tables Needed

- workflow.vworkflowwithdefinition
- workflow. process_definition

```
select process definition.name, vworkflowwithdefinition.name,
 TO_DATE('19700101000000','YYYYMMDDHH24MISS')+(start_date)/24/60/60/1000
 FROM   workflow.vworkflowwithdefinition,
 workflow.process definition
 where running='Y' and
process_definition.id=workflow.vworkflowwithdefinition.pd_id
```



Section I: Bonus Material – Mutating Tables

Lesson: Overview

Jump to TOC

Introduction

In this section, you will learn more about triggers for mutating table errors.

Section contents

Example of package to avoid the mutating table error

```
PACKAGE Pk_Inactive_Address_Change AS

  PROCEDURE P_Init_Addr_Pidm_Table;
  PROCEDURE p_add_pidm_to_list ( p_pidm NUMBER );
  PROCEDURE p_spraddr_inactvaddress;

END Pk_Inactive_Address_Change;


PACKAGE BODY Pk_Inactive_Address_Change AS

  -- structure to hold pidm numbers
  TYPE v_pidm_table_type IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  v_addr_pidm_table v_pidm_table_type;

PROCEDURE P_Init_Addr_Pidm_Table IS

BEGIN --initialize pidm table to empty
    v_addr_pidm_table.DELETE;
  END P_Init_Addr_Pidm_Table;

 /*-------------------------------------------------------*/
  PROCEDURE p_add_pidm_to_list ( p_pidm NUMBER ) IS
  /*-------------------------------------------------------*/
  BEGIN
    dbms_output.ENABLE;
    v_addr_pidm_table(NVL(v_addr_pidm_table.LAST,0) + 1) := p_pidm;
      dbms_output.PUT_line(v_addr_pidm_table(1)||' pidm ' ||p_pidm);
  END p_add_pidm_to_list;
/*-------------------------------------------------------*/
PROCEDURE p_spraddr_inactvaddress IS
/*-------------------------------------------------------*/
      v_Params          Gokparm.t_parameterlist;
      event_code        gtveqnm.gtveqnm_code%TYPE;
      fullname          VARCHAR2(50);
      p_id              spriden.spriden_id%TYPE;
      v_element         PLS_INTEGER;
      v_inactive        VARCHAR2(1);
      v_pidm                  NUMBER;
```

Example of package to avoid the mutating table error, continued

```
  BEGIN
    IF  Goksyst.f_isSystemLinkEnabled('WORKFLOW')
                  THEN
                  event_code
:=SUBSTR(Gokevnt.F_CheckEvent('WORKFLOW','INACTVADDRESS'),1,20);
                  v_element := v_addr_pidm_table.FIRST;
             LOOP
                        EXIT WHEN v_element IS NULL;
                        v_inactive := NULL;
                        BEGIN
                  SELECT spraddr_pidm INTO v_pidm FROM spraddr
                                    WHERE
spraddr_pidm=v_addr_pidm_table(v_element) AND
                                    spraddr_status_ind IS NULL;
                      EXCEPTION WHEN NO_DATA_FOUND THEN
                        v_inactive :='I';
                        END;
                        IF v_inactive ='I' THEN
                        BEGIN
                        SELECT id, F_Format_Name
(v_addr_pidm_table(v_element),'FML') INTO p_id, fullname
                                    FROM spriden
                                    WHERE
spriden_pidm=v_addr_pidm_table(v_element) AND spriden_change_ind IS NULL;
                                    EXCEPTION
                                    WHEN NO_DATA_FOUND THEN EXIT;
                      END;
                        dbms_output.PUT_line(v_addr_pidm_table(v_element)||'
'||fullname);
                        v_Params(1).param_value := 'INACTVADDRESS';
                        v_Params(2).param_value := 'SCT Banner';
                        v_Params(3).param_value := 'Address INACTIVE for - '
|| fullname || ' ID - '||id;
                        v_params(4).param_value :=
v_addr_pidm_table(v_element);
                        v_params(5).param_value := fullname;
                        v_params(6).param_value := p_id;
                        Gokparm.Send_Param_List(event_code, v_Params);
                        v_element := v_addr_pidm_table.NEXT(v_element);
                        END IF;
                  END LOOP;
      END IF;
  END p_spraddr_inactvaddress;
END Pk_Inactive_Address_Change;
```

Section J: Appendix

Lesson: Overview

Jump to TOC

Introduction

In this section, you will find more advanced information regarding Banner event definitions and Workflow parameters.

Section contents

Section J: Appendix

Lesson: Script to Port Banner Event Definition Data

Jump to TOC

Parameter_Load_Script.sql

This script will extract the event data from Banner Event tables, and create a sql script to load the data into another instance. Run the following script in the original Banner instance, and run the created script in the new Banner instance.

<COPY FROM NEXT LINE>

```
/*
This script can be run against the Event Queue Definition data in one
instance of BANNER and then used to import an event into another instance.

The output of this script should be stored with any other import data, such
as bootstraps, pl/sql procedures, table triggers and other things that will
make up your workflow and allow it to function in another DB instance.

5/3/2001 - Josh Aversa
*/
undefine &&EVENT_NAME
undefine &&GROUP_NAME
set serveroutput on
set head off
set pagesize 1000
set linesize 1000
set verify off
set echo off
set feedback off
/* Change the following name to your actual pathname */
spool C:\event_&&EVENT_NAME.sql
select '------------------------------' from dual;
select '--Parm Group Name Validation--' from dual;
select '------------------------------' from dual;
select 'insert into gtveqpg values('''||
GTVEQPG_CODE||''','''||
GTVEQPG_DESC||''','''||
GTVEQPG_USER_ID||''',to_date('''||
GTVEQPG_ACTIVITY_DATE||'''));'
FROM GTVEQPG
WHERE GTVEQPG_CODE = '&&GROUP_NAME';
select '------------------------------' from dual;
select '--Parm Name Validation      --' from dual;
select '------------------------------' from dual;
select 'insert into gtveqpm values('''||
GTVEQPM_CODE||''','''||
GTVEQPM_DESC||''','''||
```

Parameter_Load_Script.sql, continued

```
GTVEQPM_USER_ID||''',to_date('''||
GTVEQPM_ACTIVITY_DATE||'''));'
FROM GTVEQPM, GOREQPG
WHERE GOREQPG_EQPG_CODE = '&&GROUP_NAME'
  AND GTVEQPM_CODE = GOREQPG_EQPM_CODE;
select '------------------------------' from dual;
select '--Parm Group Definition Form--' from dual;
select '------------------------------' from dual;
select 'insert into goreqpg values('''||
GOREQPG_EQPG_CODE||''','''||
GOREQPG_EQPM_CODE||''','||
GOREQPG_SEQNO||',''''||
GOREQPG_DEFAULT_VALUE||''','''||
GOREQPG_TARGET_NAME||''','''||
GOREQPG_USER_ID||''',to_date('''||
GOREQPG_ACTIVITY_DATE||'''));'
FROM GOREQPG
WHERE GOREQPG_EQPG_CODE = '&&GROUP_NAME';
select '------------------------------' from dual;
select '--Event Validation Table    --' from dual;
select '------------------------------' from dual;
select 'insert into gtveqnm values('''||
GTVEQNM_CODE||''','''||
GTVEQNM_DESC||''','''||
GTVEQNM_USER_ID||''',to_date('''||
GTVEQNM_ACTIVITY_DATE||'''));'
FROM GTVEQNM, GOREQNM
WHERE GOREQNM_EQPG_CODE = '&&GROUP_NAME'
  AND GTVEQNM_CODE = GOREQNM_EQNM_CODE;
select '------------------------------' from dual;
select '--Event que name def form--' from dual;
select '------------------------------' from dual;
select 'insert into goreqnm values('''||
GOREQNM_EQNM_CODE||''','''||
GOREQNM_EQPG_CODE||''','''||
GOREQNM_EQTS_CODE||''','''||
'I'||''','''|| --I SET THE STATUS TO 'I' SO IT ISN'T ACTIVE IN THE NEW SYSTEM
GOREQNM_USER_ID||''',to_date('''||
GOREQNM_ACTIVITY_DATE||'''));'
FROM GOREQNM
WHERE GOREQNM_EQPG_CODE = '&&GROUP_NAME';
undefine EVENT_NAME
undefine GROUP_NAME
spool off
```

Section J: Appendix

Lesson: Passing Workflow Parameters To and From SQL Business Components

Jump to TOC

Overview

This document has been created to address how SunGard Workflow handles parameters that are passed to and from business components with the component type of internal. More specifically, this document will discuss how parameters handled when making Oracle PL/SQL calls.

This document is intended for internal training, but includes no material that should not be shared with clients. As an internal training document, it runs the risk of not being updated and becoming stale. Please take this into consideration before sharing with a client.

How do internal components pass parameters to PL/SQL calls?

Internal components use a strings interface to pass parameters to and from the applications they interface. This means all parameters, regardless of their data type, will be passed and received as strings. Each data type will be converted using a specific format pattern, shown in figure 1.0 below. This format will be used when passing data from Workflow to an external system and expected when data is returned from the external system to Workflow.

Figure 1.0

Workflow Parameter Formats	
Text	Passed as a string, no formatting done
Date	dd-mon-yyyy hh24:mi:ss (17-Aug-2006 15:23:00)
Numeric	String representation of number (most systems will automatically convert to a numeric field by default.
Boolean	Passed as the string ‘true’ or ‘false’

How do internal components receive parameters to applications?

Internal Components also use the strings interface when receiving parameters. The Workflow system, inspects the Internal Component’s parameter type and then applies the formatting (shown in figure 1.0) to convert the passed string to the Workflow data type. If the Workflow system cannot convert a string to the data type an alert will be raised.

Section J: Appendix

Lesson: Passing Workflow Parameters To and From SQL Business Components (Continued)

Jump to TOC

Oracle stored procedure examples

Date example

Let's take a fictitious example in which Workflow Internal Component "Find Next Birthday", calls a SQL stored procedure that takes a date as input and searches the BIRTHDAY table in Oracle for the first person's birthday that occurs on or after the passed date.

The Internal Component has three parameters:

Parameter Name	Type	Required	Guaranteed
today	date	Yes	No
nextBirthDate	Date	No	Yes
employeeIdOfNextBirthDay	Numeric	No	Yes

Pseudo code for Oracle procedure to do birthday calculation:

```
CREATE OR REPLACE PROCEDURE findNextBirthday (
      today IN VARCHAR2,
      nextBirthDay OUT VARCHAR2,
      employeeIdOfNextBirthDay OUT VARCHAR2 )
```

Notice that all parameters have been defined as VARCHAR2. This is because we are using the strings interface and every parameter passed from and to the Workflow Internal Component must be a string. We will do date conversions in the logic below.

```
IS
     procToday DATE;
     procNextBirthDay DATE;

     // convert today (passed as string) to a date for use in
     // our query
     procToday := TO_DATE( today, 'DD-MON-RRRR HH24:MI:SS' );
```

Oracle stored procedure examples – date example

Perform SQL query here to select id and birthday of the employee with the next birthday on or after the passed date. The date of birth will be selected into procNextBirthDay and the employee id will be selected into employeeIdOfNextBirthday. ** Note, Oracle will automatically convert a number into a VARCHAR for us.)

```
    // convert procNextBirthDay into a varchar to be returned to
    // workflow
    employeeIdOfNextBirthday :=
                TO_CHAR( procNextBirthDay, 'DD-MON-RRRR HH24:MI:SS');

END;
```

Boolean example

When setting up database columns to store Boolean values, it is common practice to use a VARCHAR2(1) or CHAR(1) with a 'Y' = true and 'N' = false or a numeric field with a 0 = true and a 1 = false. The following example assumes one of these two setups in the database.

For this example let us take a look at a fictitious database procedure that will look at a passed date and determine if any employees were born on that date. We will add a second parameter to the function that if 'true' will look to see if any employees were born within five days of the passed date. If any records are found the procedure will return true.

The Internal Component has three parameters:

Parameter Name	Type	Required	Guaranteed
today	Date	Yes	No
checkFullWeek	Boolean	Yes	No
employeeFound	Boolean	No	Yes

Section J: Appendix

Lesson: Passing Workflow Parameters To and From SQL Business Components (Continued)

Jump to TOC

Oracle stored procedure examples – boolean example, continued

Pseudo code for Oracle procedure to do birthday calculation:

```
CREATE OR REPLACE PROCEDURE findBirthday
      today IN VARCHAR2,
      checkFullWeek IN VARCHAR2,
      employeeFound OUT VARCHAR2 )
```

Notice that all parameters have been defined as VARCHAR2. This is because we are using the strings interface and every parameter passed from and to the Workflow Internal Component must be a string. We will do date conversions in the logic below.

```
IS
     procToday DATE;
     procCheckFullWeek VARCHAR2(1);

     // convert today (passed as string) to a date for use in
     // our query
     procToday := TO_DATE( today, 'DD-MON-RRRR HH24:MI:SS' );

     // convert checkFullWeek to the database representation of
     // a boolean
     procCheckFullWeek := decode( checkFullWeek, 'true', 'Y',
                                                 'false', 'N' );


     /
     /
     c
```

Perform SQL query here to determine if any employees can be found having a birth date equal to the date passed or a birthday within 5 days of the date passed if chekFullWeek is set to true. Result of the SQL query will return an integer, results, equal to the number of records found.

```
     // convert numeric variable 'results' to a varchar with the
     // correct Boolean formatting

     IF results > 0 THEN employeeFound := 'true';
     ELSE employeeFound := 'false';
     END IF;

END;
```

Section J: Appendix

Lesson: Passing Workflow Parameters To and From SQL Business Components (Continued)

Jump to TOC

Doing a conversion within a PL/SQL call example

In the previous two examples we showed how parameters could be converted within the body of a PL/SQL procedure. In this example, we will show how you can convert a Workflow parameter from its string representation to an Oracle data type within the PL/SQL call.

*** Please note that this only pertains to parameters being passed from Workflow to an Oracle PL/SQL procedure. Parameters being returned from Oracle still must be converted to the proper Workflow string representation within the Oracle PL/SQL procedure.

Date example -- revisited

Using the date example above, assume we change the "IN" parameter "today" from a VARCHAR2 to a DATE.

```
CREATE OR REPLACE PROCEDURE findNextBirthday (
      today IN DATE,
      nextBirthDay OUT VARCHAR2,
      employeeIdOfNextBirthDay OUT VARCHAR2 )
```

To execute this procedure from Workflow, we will need to change our calling Business Component's "Client Launch Parameter".

Old Launch Parameter

procedure=findNextBirthday(@today)

New Client Launch Parameter

procedure=findNextBirthday(to_date(@today, 'DD-MON-RRRR HH24:MI:SS'))

Section J: Appendix

Lesson: Passing Workflow Parameters To and From SQL Business Components (Continued)

Jump to TOC

Doing a conversion within a PL/SQL call example, continued

Boolean example -- revisited

Using the Boolean example above, assume we change the “IN” parameter “today” from a VARCHAR2 to a DATE. Also assume we keep the “IN” parameter “checkFullWeek” as a VARCHAR2, but now assume a “Y” or an “N” will be passed to our PL/SQL procedure instead of “true” or “false”.

```
CREATE OR REPLACE PROCEDURE findBirthday
      today IN DATE,
      checkFullWeek IN VARCHAR2,
      employeeFound OUT VARCHAR2 )
```

Old Client Launch Parameter

procedure=findNextBirthday(@today, @checkFullWeek)

New Client Launch Parameter

procedure=findNextBirthday(to_date(@today, ‘DD-MON-RRRR HH24:MI:SS’), decode(checkFullWeek, ‘true’, ‘Y’, ‘false’, ‘N’))

Oracle SQL query examples

For Oracle SQL queries made using the "SQL Query" data type in Workflow, the rules outlined in Section 4 also apply. When executing a SQL query, however, it is important to remember that your query result must be a text string formatted to represent the data type of the Business Component parameter to which it will be mapped.

Date example

Assume we wanted to run a simple SQL query from a Workflow Business Component that selects the sysdate from Oracle and populates it into our component parameter "today", which has been configured as a date. Because the parameter "today" is a date, our return values must be a string of format "DD-MON-RRRR HH24:MI:SS. Our Client Launch Parameter to execute this query should look as follows:

```
sqlQuery=select to_char(sysdate, 'DD-MON-RRRR HH24:MI:SS') into @today from dual
```

The query above shows a select statement that pulls data out of the database and into the Workflow system. If we are to change this query such that a Workflow date parameter is passed to database as part of the select statement, we will need to convert the string representation to an Oracle date. Assume we want to select the id from the 'INVOICE' table where the 'ACTIVITY_DATE' is equal to the date passed in the Workflow parameter 'query_date'.

```
sqlQuery=select id into @id from INVOICE where INVOICE.CHANGE_DATE = to_date(@query_date,
'DD-MON-RRRR HH24:MI:SS')
```

Boolean example

Assume we want to select a value representing a Boolean from a table in Banner. For this example pretend we have a table "USER" with field "ACTIVE", which is a VARCHAR2(1). If "ACTIVE" has a value of 'Y' the user account is active. If "ACTIVE" has a value of 'N' the user account is inactive. Our Business Component has two parameters; "id" (id of a Banner user) and "active" (Boolean to store the result of our query). When formulating this query we must remember that Workflow is expecting a string of format "true" or "false" to be returned. Our Client Launch Parameter to execute this query should look as follows:

```
sqlQuery=select decode(ACTIVE, 'Y', 'true', 'N', 'false),'') into @active from USER where USER.ID =
@id
```

Release Date

Jump to TOC

This workbook was last updated on 1/28/2008.