Process Scheduler Request		×			
Operator ID: PS Run (	Operator ID: PS Run Control ID: #USER005				
- Run Location	Output Destination	]			
Client C Server	File     Printer     Window				
Server:	File/Printer: %temp%\	ОК			
Run Date/Time:	Run Recurrence	Cancel			
Date: 01/22/99	Once				
Time: 07:21:00 PM	Name:				
Reset to current Date/Time	New Update Delete				
Description	Name Process Type Descr	-			
Application Engine Application Engine	AEADHOC Application Engine PTPEMAIN COBOL SQL				

Figure 40.32 Submitting a Process Scheduler request

Request Messages	nites Use Process Inquire	•3 2		+ / /		
Application: AE USER005	The control to .			cess Instance		
Time Severity	Message					
19.22.10 10	1 Requests found for PS.#US	ER005			14 14	
19.22.11 10	Executing request 1 of 1				-	
19.22.11 10	Beginning Application AE.US	ER005 U	Iser Application 0	05	-	
19.22.18 10	BOND_LOG contains 471 red	cords				
19.22.22 10	DED_CALC contains 90 reco	rds			-	
19.22.25 10	DED_LINE contains 29 recor	ds				
19.22.32 10	ESPP_RUNCTL contains 2 re	ecords			*	
Process Instance: 31				Messages		Update //

Figure 40.33 Reviewing Process Request messages

We are once again successful! The message log output matches that of our SQR version. Only tables that contain rows of data are displaying. Using the DO When construct, we've filtered out all tables with a zero row count. Two additional statement types—the DO Until and DO While statement types—control section execution in a similar fashion.

# 40.2 SOR/APPLICATION ENGINE COMPARISON

Once again, let's take a look at the logical structure of both our programs:

SQR:

Application Engine:

Begin-Program	USER003
User prompted	Cache assignment
Main-Step1	MAIN.STEP1
Count-Step1	COUNT.STEP1
Msg-Step1	COUNT.STEP2
	MSG.STEP1

This time the structures are slightly different. The SQR program doesn't need an additional step to perform a decision. A simple IF statement is used to determine if the Msg-Step1 procedure should be performed. Application Engine requires the additional step to build a DO When condition. Based on the results, the MSG section is performed.

### KEY POINTS

- 1 You can control the processing logic using a DO When, DO While, or DO Until statement. This adds decision-making capability to your program and can regulate which sections are performed.
- 2 The PSLOCK table is often used as a dummy table to evaluate &BIND data values. It can be used in the same manner as the DUAL table is used in Oracle.



Dynamic sections

41.1 Exercise 6: Calling dynamic sections 871

41.2 SQR/Application Engine comparison 886

41.3 Dynamic sections in PeopleSoft 886

An Application Engine program has the ability to call a section dynamically. This is a very powerful feature. Sections may be created and, based on certain conditions, a particular section may be executed. Let's begin.

### 41.1 EXERCISE 6: CALLING DYNAMIC SECTIONS

Our exercise is simple. We are going to dynamically call a section that either writes the message "Hello World" or "Goodbye." Dynamic sections are called based on the contents of the cache field AE\_SECTION. This field must exist in the cache record we've designated for our application. We'll populate this field on the Process Request panel with the name of the section we'd like to perform. We begin by displaying a simple SQR that prompts the user for their choice of messages to display. Keep in mind the SQR version isn't dynamic—it simply performs the routine based on user selection—but it will demonstrate the logic flow as if it were dynamic.

### 41.1.1 Creating an SQR version

```
! USER006.SOR
begin-program
input $choice 'Enter Section# (1=Hello 2=Goodbye)' maxlen=1
evaluate $choice
   when = '1'
     do Hello-Step1
   when = '2'
     do Goodbye-Step1
end-evaluate
end-program
begin-procedure Hello-Step1
show 'Hello World'
end-procedure
begin-procedure Goodbye-Step1
show 'Goodbye'
end-procedure
```

If the user enters a '1', the SQR.log looks like this:

Hello World

If the user enters a '2', the SQR.log looks like this:

Goodbye

Let's create a version of the program using Application Engine.

We begin by adding the USER006 application name and section MAIN (figure 41.1).

*Navigation:* Go  $\rightarrow$ PeopleTools  $\rightarrow$ Application Engine  $\rightarrow$ Use  $\rightarrow$ Application Engine  $\rightarrow$ Application  $\rightarrow$ Add

Product: PS/AE  Application: USER006	OK
	Cancel
Section: MAIN	
Database Platform:	

Figure 41.1 Naming the application Fill in the description, version, and message set number as we've done in the past exercises. We're going to use a different cache record called AE\_TESTAPPL\_AET. This is a delivered PeopleSoft record. The field AE\_SECTION is contained in this record so it's perfectly suited for our dynamic section exercise. We could have added this field to our USER\_AET cache record, but I wanted to demonstrate the cache record assignment. There is no need to create additional cache records if one exists that meets your requirements.

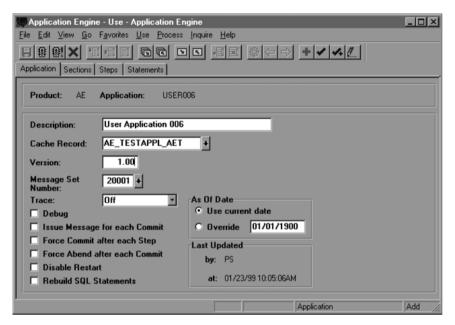


Figure 41.2 Defining the application

We add our section MAIN description (figure 41.3).

Application Engine - Use - Application Engine File Edit View Go Favorites Use Process Inquire Help G G M VI VIIII - Constant Co	
Product: AE Application: USER006 Section:	MAIN Platform:
Effective Date:       01/23/1999       Status:       Active         Description:       USER006 Main         © Commit after each step         Type       •         • Preparation Only         • Critical Database Updates         Actions         New       Save As         Rename       Delete	
	Sections Add

Figure 41.3 Defining section MAIN

We call our step STEP1 and click on the DO section radio button. Next, you click on the Edit button to indicate the section to perform.

Application Engine - Use -	Application Engine					_ 🗆 🗵
<u>File Edit View Go</u> Favorites	<u>U</u> se <u>P</u> rocess <u>I</u> nqui	re <u>H</u> elp				
			⇒ + ✓	4		
Application Sections Steps S	Statements					
Product: AE Applicati	ion: USER006	Section: MAIN	DB	Pltfrm:		
Effective Date: 01/23/	1999	Ca	che Record:	AE_TESTA	APPL_AET	-
Step: STEP1 S	tatus: Active	•	Step Pos	ition		-
O SQL Statement	Edit		×	± Ŧ	×	
O Mass Change		+				
DO Section	Edit		DO E	dit	Exists	
C COBOL Program			D	O Select		
	Jnsuccessful	DO Method		0 When		
	Abort Application	Select and F	etch [	0 Until		
C After this Step C C Later C		C Re-Select	D	0 While		
	Suppress Error				1	
			Steps			Add //

Figure 41.4 Defining STEP1

### 41.1.2 The &SECTION symbolic

When the DO Section Properties box appears, click on the dynamic DO checkbox (figure 41.5). Notice the &SECTION symbolic appears. You can also notice the product, application, and section edit boxes have been grayed out. This means the section you want to perform must exist within your Application Engine program. You cannot dynamically call a section from another Application Engine program.



When you return to the Step Definition panel, you'll notice the section being called is set to (DYNAMIC). When this step is executed, the DO section is determined by substituting the contents of the AE\_SECTION cache field.

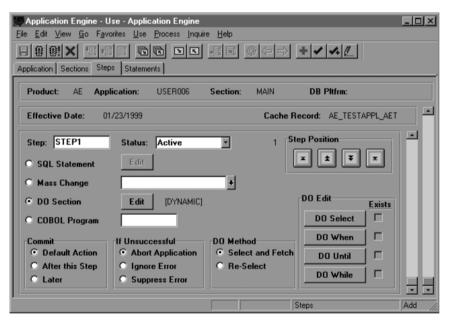


Figure 41.6 The DO section has been completed

We'll now add another section, called HELLO, to our application (figure 41.7).

1	Add Application E	ngine	×	
	Product:	PS/AE	ОК	
	Application:	USER006	Cancel	
	Section:	HELLO		<b>Figure 41.7</b>
	Database Platform	x X		Figure 41.7 Adding another section
				called HELLO

Fill in the description for the HELLO section (figure 41.8).

Application Engine - Use - Application	ation Engine		_ 🗆 🗵
<u>File Edit View Go Favorites Use</u>	<u>P</u> rocess <u>I</u> nquire <u>H</u> elp		
			.1
Application Sections Steps Statemer			- 1
Product: AE Application:	USER006 Section:	HELLO Platform:	
Effective Date: 01/23/1999	Status: Active	•	
Description: USER006 He	llo ₩orld		
Commit after each step			
Туре			
Preparation Only			
C Critical Database Updates			
Actions			
	Rename Delete		
New 2946 AS	Tename Delete		
1			<u> </u>
		Sections	Add //

Figure 41.8 Defining section HELLO

We call the first step of the HELLO section STEP1 (figure 41.9).

Application Engine - Use - Application Engine File Edit View Go Favorites Use Process Inquire Process Inquire Application Sections Steps Statements		_ 🗆 🗙
Product: AE Application: USER006	Section: HELLO DB Pltfrm:	
Effective Date: 01/23/1999	Cache Record: AE_TESTAPPL_AET	-
Step:     STEP1     Status:     Active       ©     SQL Statement     Edit       ©     Mass Change       ©     DD Section	▼ 1 Step Position	
COBOL Program Commit Default Action After this Step Later Commit Suppress Error Commit Commit Suppress Error Commit Commi	D0 Select  D0 When  Select and Fetch B0 White D0 White	× ×
	Steps	Add //.

Figure 41.9 Defining STEP1 of section HELLO

We use the & MSG function to display "Hello World" on the message log (figure 41.10).

Application Engine - Use - Application Engine         File       Edit       View       Go       Favorites       Use       Process       Inquire       Help         Image: Sections       Steps       Statements       Image: Statements	
Product: AE Application: USER006 Section: HELLO DB Pltfrm:	
Effective Date: 01/23/1999	-
Step: STEP1	-
Type: U + Update/Insert/Delete Re-Use Ret	um
&MSG(,1,'Hello World')	
Statements	Add //

Figure 41.10 Adding the &MSG statement text

EXERCISE 6: CALLING DYNAMIC SECTIONS

We'll now add another section to our application called GOODBYE (figure 41.11).

Add Application	Engine		×
Product: Application:	PS/AE USER006		OK Cancel
Section:	GOODBYE		Lancei
Database Platfo	orm:	•	

Figure 41.11 Adding another section called GOODBYE

Fill in the description for the GOODBYE section (figure 41.12).

	ne - Use - Applic F <u>a</u> vorites Use	Process Ing		• <b>(+</b>  -) •	× × /	
Product: AE	Application:	USER006	Section:	GOODBYE	Platform:	
Effective Date: Description: Commit after Type Preparation Critical Data Actions New	Only abase Updates		Active			×
				Sec	tions	Add //.

Figure 41.12 Defining section GOODBYE

We call the first step of the GOODBYE section STEP1 (figure 41.13).

Application Engine - Use - Application Engine File Edit View Go Favorites Use Process Inquir	• <u>H</u> ep ■⊠ 廖 Þ → <b>+ / </b> ▲ 【	_ 🗆 ×
Application Sections Steps Statements		
Product: AE Application: USER006	Section: GOODBYE DB Pltfrm:	
Effective Date: 01/23/1999	Cache Record: AE_TESTAPPL_AET	
Step:     STEP1     Status:     Active       • SQL Statement     Edit	Step Position	
C Mass Change     D0 Section     Edit     COBOL Program	DO Edit Exists	
Commit © Default Action © After this Step © Later © Suppress Error	DD Method © Select and Fetch © Re-Select DD While	
	Steps	Add //

Figure 41.13 Defining STEP1 of section GOODBYE

We use the &MSG function to display "Goodbye" on the message log (figure 41.14).

	_	ne - Use - Applic					_ D ×
Eile Edit V		Favorites Use				• • •	
Application	Sections	Steps Stateme	ents				
Product	: AE	Application:	USER006	Section:	GOODBYE	DB Pltfrm:	
Effectiv	e Date:	01/23/1999					-
Step:	STEP1						-
Туре:	U +	Update/Insert/D	elete		🗆 Re-Use	Return	
&MSG(,1,	'Goodbye')					1	
							ادادا
					0	in a second	
					Sta	tements	Add //,

Figure 41.14 Adding the &MSG statement text

EXERCISE 6: CALLING DYNAMIC SECTIONS

We're ready to test the USER006 application.

Return to the Process Request panel and add the Run Control ID #USER006. Click the OK button.

*Navigation:* Go  $\rightarrow$ PeopleTools  $\rightarrow$ Application Engine  $\rightarrow$ Process  $\rightarrow$ Request  $\rightarrow$ Add

스	
OK Cancel	Figure 41.15 Adding the Run Control ID

### 41.1.3 The AE\_SECTION cache field

When the Process Request panel appears, click on the Field edit box. and scroll through the field list (figure 41.16). Select the cache field AE\_SECTION.

Valid Values		X
Field Name	Descr	OK
AE_INT_15	Number Work Field	
AE_INT_2	Number Work Field	Cancel
AE_INT_3	Number Work Field	
AE_INT_4	Number Work Field	
AE_INT_5	Number Work Field	
AE_INT_6	Number Work Field	
AE_INT_7	Number Work Field	
AE_INT_8	Number Work Field	
AE_INT_9	Number Work Field	
AE_PRODUCT	Product	
AE SECTION	Section	
AE_STEP	Name	
PROCESS_INSTANCE	Process Instance	1
<b>x</b>	1	

Figure 41.16 The cache field drop-down list box

We can now enter the name of the section which we'd like to perform. For the AE\_SECTION cache field, we assign a value of HELLO. Our USER006 Application Engine program substitutes the section HELLO when it processes the &SECTION symbolic. Once the Process Request panel is populated correctly, click on the Traffic Signal to initiate a Process Scheduler request.

	cess - Request es ⊥se Process Inquire Help ■ © © ● ● ● ● ● ● ● ● ● ● ● ● ●	
Operator Id: PS	Run Control ID: #USER006	
Request Number: Process Frequency: Product: Application:	1 Always AE + USER006 + User Application 006	
Fields Field AE_SECTION	Value I HELLO	
	Request	Update //

Figure 41.17 Assigning the dynamic section field the section HELLO

Eik			rocess - Request rites Use Process Inquire		<u> </u>	+ 🗸 🔏 🥂		
	Operator:	PS	Run Control ID:	#USER0	06			
	Application: AE USER00		v Messages O Trace		Use Latest Pro Use Previous	cess Instance		
	Time	Severity	Message					
	10.17.59	10	1 Requests found for PS.#US	SER006			-	
	10.18.00	10	Executing request 1 of 1				*	
	10.18.00	10	Beginning Application AE.US	ER006 L	User Application 0	06		
	10.18.02	10	Hello World				-	
	10.18.02	10	Application AE.USER006 end	ded norm	ally			
Pro	cess Instance:	32				Messages		Update //

Figure 41.18 Reviewing process request messages

After examining the message log, you can see the HELLO section was performed. This was caused by populating the AE\_SECTION cache field with the section which you'd like to perform.

Let's test our program again using another section. Return to the Process Request panel and assign the value GOODBYE to the AE\_SECTION cache field (figure 41.19). Execute the program again and go to the message log to view the results.

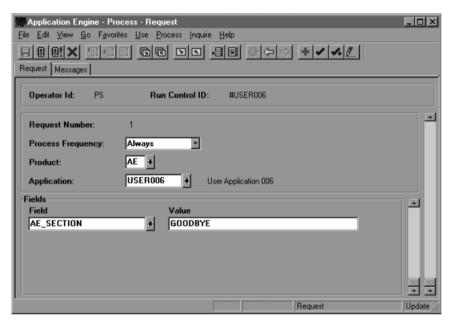


Figure 41.19 Assigning the dynamic section field the section GOODBYE

This time the GOODBYE section was performed (figure 41.20).

		rocess - Request> rites Use Process Inquire Help ■ ■ ■ ■ ■ ■ ● → ● ● ● ● ● ●
Operator:	PS	Run Control ID: #USER006
Application: AE USER006	View	Messages C Trace C Use Latest Process Instance
Time	Severity	Message
10.18.46	10	1 Requests found for PS.#USER006
10.18.46	10	Executing request 1 of 1
10.18.46	10	Beginning Application AE.USER006 User Application 006
10.18.48	10	Goodbye
10.18.48	10	Application AE.USER006 ended normally
ocess Instance: 3		Messages Update

Figure 41.20 Reviewing process request messages

#### 41.1.4 Multiple process requests

Let's try another quick experiment. We can use the Process Request panel to run multiple requests at once. We could have easily run the HELLO and GOODBYE versions of our exercise one after the other in the same run request.

Let's start by creating a new process request under a new Run Control ID.

Since we're running multiple requests, we use the Run Control ID 'MULTIPLE' (figure 41.21). Now we need to populate the Process Request panel.

Navigation: Go ->PeopleTools ->Application Engine ->Process ->Request ->Add

Add Request	×	
Run Control ID: MULTIPLE	ОК	
		ure 41.21 ling the Run Control ID

So far nothing seems different (figure 41.22). The dynamic section HELLO will be executed. Take a look at the outermost scroll bar on the right. Each process request we enter can be viewed using the outer scroll bar. We can add another process request by placing the cursor in one of the outer scroll fields (process frequency, product, or application) then pressing the F7 key to insert a new row.

	ocess - Request ites Lise Process Inquire Help 티 한다 보도 물론 중승규가 부탁하겠	
Operator Id: PS	Run Control ID: MULTIPLE	
Request Number:	1	-
Process Frequency:	Always	
Product:	AE +	
Application:	User Application 006	
Fields		
Field	Value	
AE_SECTION	+ HELLO	
	Request	Add

Figure 41.22 Adding Request Number 1 using the HELLO dynamic section

After pressing the F7 key, a new row can be filled in with our second set of run parameters. In figure 41.23, we have selected the dynamic section GOODBYE. Notice the request number for the GOODBYE section is incremented to 2. When we submit our request, the HELLO request will be executed followed by the GOODBYE request. Let's run the request and look at the message log.

Figure 41.24 displays the Messages panel. The first line shows that two requests were found for our run. The first request is executed (1 of 2) and displays the "Hello World" message. The second request is then executed (2 of 2) and displays the "Goodbye" message (though not visible in the screen shot). We could have executed all of our exercises consecutively in one process request run.

	access - Request Res Lise Process Inquire Help	
Operator Id: PS	Run Control ID: MULTIPLE	
Request Number: Process Frequency: Product: Application:	2 Always AE + USER006 + User Application 006	-
Fields Field AE_SECTION	Value + GOODBYE	× -
	Request	Correct //

Figure 41.23 Adding Request Number 2 using the GOODBYE dynamic section

<u>F</u> ile <u>E</u> dit ⊻iew		Yrocess - Request □ orites Use Process Inquire Help □□ □□ □□ □□ □□ ● ● ● ● ● ● ● ● ● ● ● ●	1×				
Operator: Application: AE USER008	PS View	Run Control ID:       MULTIPLE         w       © Use Latest Process Instance         Messages       © Trace         © Use Previous       ¥	-				
Time	Severity	Message					
09.38.15	10	2 Requests found for PS.MULTIPLE					
09.38.15	10	Executing request 1 of 2					
09.38.16	10	Beginning Application AE.USER006 User Application 006					
09.38.18	10	Helio World					
09.38.18	10	Application AE.USER006 ended normally					
09.38.18	10	Application AE. USER006 ended normally					
09.38.19	10	Beginning Application AE.USER006 User Application 006	-				
Process Instance:	70	Messages	ect //,				

Figure 41.24 Examining the message log for the multiple request run

# 41.2 SOR/APPLICATION ENGINE COMPARISON

If we look at the logical structure of both our programs now

SQR: Application Engine: Begin-Program User prompted Main-Step1 When 1 Hello-Step1 When 2 Goodbye-Step1

We can see the Application Engine program is much more streamlined. The section is assigned on the Process Request panel and used in place of the &SECTION symbolic.

# 41.3 DYNAMIC SECTIONS IN PEOPLESOFT

You may be wondering where you can find an example of dynamic sections in an existing PeopleSoft application. A perfect example would be the payment predictor process, called PREDICT, found in Accounts Receivable. Its purpose is to match incoming payments with the associated items (or invoices). Several delivered sections or algorithms exist that can be selected to match payments based on certain criteria. A special payment predictor setup table is used to store the algorithm name to be used. When PREDICT is run, the setup information is retrieved. The algorithm is assigned to the AE\_SECTION cache field and substituted for the &SECTION symbolic. Using dynamic sections in this manner provides a great deal of flexibility.

### KEY POINTS

- 1 Sections are called dynamically when the value found in the AE\_SECTION bind variable is substituted as the section represented by the &SECTION symbolic.
- 2 Dynamic sections can be found in several PeopleSoft Applications such as Payment Predictor. The use of dynamic sections allows the user to tailor programs to meet their own business requirements.
- **3** Multiple process requests can be submitted in one execution run. This is useful when you would like processes to run consecutively.



Using Run Controls-part A

- 42.1 Exercise 7: Delete process definitions 888
- 42.2 Build a new Run Control record 889
- 42.3 Building the Run Control panel 893
- 42.4 Create a new panel group 897
- 42.5 Attaching the panel group to a menu 899
- 42.6 Assigning operator security 900
- 42.7 Testing the new panel 902
- 42.8 Creating our process definition 903

The exercises we have completed thus far were designed to demonstrate the capabilities of Application Engine. You may not have a need to display a message saying "Hello World." You may not need any of the programs we've created! The important thing is that you've learned the concepts behind Application Engine development. We're now ready to produce something of value. In chapter 28, we mentioned that PeopleSoft does not provide a tool to delete obsolete process definitions. You can only delete these outside of PeopleSoft using your native SQL tools. Let's create a utility to accomplish this using Application Engine. This is the perfect opportunity to introduce Run Control records in Application Engine. In order to implement this utility, we're going to go through the complete cycle of program development. Let's get started.

### 42.1 EXERCISE 7: DELETE PROCESS DEFINITIONS

Let's refresh our memory first. In chapter 28, we listed the SQL statements necessary to clean up the process definition tables.

Let's look at the SQL statements used to remove the MYPROB01 SQR Report process definition from all associated tables:

```
DELETE
  FROM PS_PRCSDEFN
 WHERE PRCSNAME = 'MYPROB01'
   AND PRCSTYPE = 'SOR Report';
DELETE
  FROM PS PRCSDEFNGRP
WHERE PRCSNAME = 'MYPROB01'
  AND PRCSTYPE = 'SOR Report';
DELETE
  FROM PS PRCSDEFNPNL
WHERE PRCSNAME = 'MYPROB01'
  AND PRCSTYPE = 'SQR Report';
DELETE
  FROM PS PRCSDEFNXFER
WHERE PRCSNAME = 'MYPROB01'
  AND PRCSTYPE = 'SQR Report';
DELETE
  FROM PSPRCSROST
 WHERE PRCSNAME = 'MYPROB01'
   AND PRCSTYPE = 'SQR Report';
DELETE
  FROM PSPNLFIELD
 WHERE PRCSNAME = 'MYPROB01'
   AND PRCSTYPE = 'SQR Report';
```

In our new process, we'll allow the user to enter the PRCSNAME and PRCSTYPE on a new Run Control panel. The Application Engine process we develop will remove the process definition from the six tables listed. Notice the first four tables use the standard prefix 'PS\_' while the last two tables do not. We'll make our program interesting by accounting for this in our program.

Before we proceed, let's take a look at the development requirements. When developing applications, this is a critical and often overlooked step. Let's go over the steps we need to take to produce our application:

CHAPTER 42 USING RUN CONTROLS—PART A

### 42.1.1 Application development steps

- create a custom Run Control record
- add PeopleCode to the Run Control record
- create a custom panel
- create a custom panel group
- attach the panel group to a menu
- assign operator security to the menu item
- create a process definition for our Application Engine program
- create the Application Engine program
- test our new application

We have a lot of work ahead of us. Fortunately, this will be a fairly easy task using PeopleTools.

One thing I'd like to resolve now is the name of our Application Engine program. This means the combination of product and application ID. We will use the product A/E (as we've done in all our exercises) and the name of the application is going to be MYPRCSDL.

Let's start by building a new Run Control record.

# 42.2 BUILD A NEW RUN CONTROL RECORD

Application Engine programs use a primary Run Control record called AE\_REQUEST. You may not have realized it at the time, but this is the underlying Run Control record we've been using when testing our applications in exercises 1 through 6. We're going to create a new custom Run Control record that will be linked (as a child record) to the AE\_REQUEST record. We start by cloning the AE\_REQUEST record (figure 42.1).

Our Run Control record only needs five fields. Of course, the record needs the standard AE\_REQUEST keys, which are OPRID, RUN\_CNTL\_ID, and REQUEST\_NBR. In addition, we want the user to enter the process type and process name. We'll now clone the AE\_REQUEST record for our purposes. Open the record AE\_REQUEST and remove all fields except OPRID, RUN\_CNTL\_ID and REQUEST\_NBR. Next, add the fields PRCSTYPE and PRCSNAME. The result should look like figure 42.1. Be careful not to save the record using the AE\_REQUEST name! We save it under a new name: MY\_RUN\_CNTL\_AE.

#### *Navigation:* Go $\rightarrow$ PeopleTools $\rightarrow$ Application Designer $\rightarrow$ File $\rightarrow$ Open $\rightarrow$ Record $\rightarrow$ AE\_REQUEST

Application Designer - Untille	<u>I</u> ools <u>G</u> o F <u>a</u> vorites <u>W</u> indow					_ 🗆 × _ & ×
Unitled	Field Name OPRID RUN CNTL_ID HEQUEST_NBR PROCESS_FREQUENCY AE_PROCESS_STATUS PROCESS_SINSTANCE PROCESS_ORIG GOTO_PROCESS_STAT LAST_RUN_DTM AE_PRODUCT AE_APPL_ID	Type Len Char 8 Char 30 Nbr 4 Char 1 Char 1 Char 1 Char 1 Char 1 Char 1 Char 1 Char 2 Char 2 Char 8	Format IH Mixed Mixed Upper Upper Upper Upper Upper Upper	When Status Instance Prcs Orig Prcs Stat	Long Name Operator Id Run Control ID Request Number Process Frequency Process Instance Process Origin Go To Process Status Last Run On Product Application	
<u>וווווווווווווווווווווווווווווווווווו</u>	s ≿ teats ≿ status ∕					NUM

Figure 42.1 Cloning the AE\_REQUEST Run Control record

Save As	X
Save <u>N</u> ame As: MY_RUN_CNTL_AE	ОК
MT_HON_CNTC_AE	Cancel

Figure 42.2 Saving our Run Control record

Use File  $\rightarrow$ Save As to save the record under a new name. Figure 42.2 shows the prompt box filled in with our new name. Let's add some underlying edit prompts for our two new fields.

Figure 42.3 displays the record in Edit View. We've added the PRCSTYPE\_VW and PRCSDEFN records as edit tables for the PRCSTYPE and PRCS-NAME fields. This allows the user to select the process

type and process name on the Run Control panel using drop-down lists.

Also, note the record keys for our new Run Control record are OPRID, RUN\_CNTL\_ID, and REQUEST\_NBR. The key attributes were copied when we saved the AE\_REQUEST record under our new record name.

Because we plan to integrate our Run Control record with the AE\_REQUEST record, we have to resolve a couple of issues. This will become clearer when we create the panel. For now, let's look at the required fields on the AE\_REQUEST record.

Application Designer - Untitled         Ele       Edt       Yiew       Inset       Build         Ele       Edt       Yiew       Inset       Build	<u>I</u> ools <u>G</u> o F <u>a</u> vorites <u>W</u> indo						_ 🗆 ×
Unitled	Field Name OPRID RUN_CNTL_ID REQUEST_NBR PRCSTYPE PRCSTYPE PRCSTYPE PRCSTYPE PRCSTYPE	Type Rec Char Yes Char Yes Nbr Yes Char Yes Char Yes	Prompt	Prompt Table OPRID_VW PRCSTYPE_VW PRCSDEFN	Set Control Field	No No No No	PeopleCo Yes No Yes No No
王王 Tel Star Tel Star Television 入 Upped , Ready	h tank h toles					NUM	×

Figure 42.3 Adding edit prompts to the process type and process name fields

Figure 42.4 displays the required fields on the AE\_REQUEST record. The first three are the record keys and will be filled in with the OPRID, RUN\_CNTL\_ID, and REQUEST\_NBR. The last two fields, AE\_PRODUCT and AE\_APPL\_ID, are also required. We plan to use the AE\_REQUEST record as the primary record on our panel with our new custom record placed in the panel as a child record. We will not be able to save the record without a product or application ID. This is an easy customization using PeopleCode. We'll simply initialize these two fields with the product and application ID we're going to use. We've already decided to use AE.MYPRCSDL as the product and application ID. Let's add some PeopleCode.

Figure 42.5 shows the PeopleCode to populate the required fields AE\_PRODUCT and AE\_APPL\_ID in the AE\_REQUEST record.

Application Designer - Untitled - A File Edit View Insert Build Iools					_ 🗆 🗙
	TIVAE_REQUEST (Record Field Name OPRID RUN_CNTL_ID REQUEST_NER PROCESS_STATUS PROCESS_STATUS PROCESS_INSTANCE PROCESS_ORIG GOTO_PROCESS_STAT LAST_RUN_DTM AE_PRODUCT AE_APPL_ID	D Type Req Edit Char Yes Char Yes Char No Char No Xlat Char No Xlat Char No Xlat Char No Xlat Char No Xlat Char No No No No No No No No No No	Prompt Table OPRID_VW AE_APPL_TBL	Set Control Field	
○○○○ Ball 入 Fed Glast References 入 Upgable 入 Fee Ready	ah ) Vétas/				×

Figure 42.4 Looking at the AE\_REQUEST required fields

Eile Edit View Insert Build	d - [MY_RUN_CNTL_AE [Record PeopleCode]] Iools <u>Go</u> Favorites <u>Window Help</u>		_ D ×
	OFRID AE_REQUEST.AE_RPODUCT = "AE"; AE_REQUEST.AE_APPL_ID = "MYPRCSDL";	I RowInit	<u> </u>
<u>&lt;1≥N</u> B& A red dect hereway & tigets Ready	A Taulis & Values		* *

Figure 42.5 Updating AE\_REQUEST record with our product and application ID

Build the current object using SQL create. This will create the table at the database level. We'll now make some modifications to our cache record, USER\_AET.

### 42.2.1 Modify our existing cache record

The cache record USER\_AET is the same record we've used for most of our exercises. We can re-use this by adding the additional fields we need for our application. We need to add the process type and process name fields passed from the Run Control record. We'll also add the AE\_SECTION field to allow us to execute sections dynamically. We'll explain this as we develop the application.

Figure 42.6 shows the modifications we've made to the USER\_AET cache record. As you can see, the three fields have been added. Now, we need to build the current record object using SQL create. We can now build the Run Control panel.

Application Designer - Untitled - [	USEB AFT (Becord)]					- 0 ×
Eile Edit View Insert Build Ion		Help				_8×
DBEG B X BB		78				
🕄 Untitled	Field Name PROCESS_INSTANCE	Type Len Nbr 10	Format  H	Short Name	Long Name Process Instance	
	COUNTER	Sign 5 Char 15	Upper	Counter Record	Scroll Count Record (Table) Name	
	FIELDNAME AE DECIDE	Char 18 Char 1	Upper Upper	Field Name	Field Name I've decided to	
	PRESTYPE	Char 30 Char 8	Mixed	Ргос Туре	Process Type Process Name	
	AE_SECTION	Char 8 Char 8	Upper	Section	Section	
- X D						
	L					
						E
						-
<u>∢</u> [▶]   Bull λ Fiel Clipst Ruferinces λ Upgrade λ Ro	ualts Xalitae					
Ready					NUM	

Figure 42.6 Modifying the USER\_AET cache record

# 42.3 BUILDING THE RUN CONTROL PANEL

We can clone an existing panel used specifically for Application Engine. The panel name is AE\_REQUEST. We used this panel when testing our applications from previous exercises. We're going to remove most of the fields on the panel, then add the new ones from our custom Run Control record.

BUILDING THE RUN CONTROL PANEL

Figure 42.7 shows the AE\_REQUEST panel with all the fields intact. Let's remove all fields except operator ID, Run Control ID, and request number. We also will keep the rightmost scroll bar. Then we'll add the Process Type and Process Name fields from our custom Run Control record. We'll make a few slight adjustments that may seem a bit odd at first, but we'll explain as we go.

*Navigation:* Go →PeopleTools →Application Designer →File →Open →Panel →AE\_REQUEST

Application Designer - Untitled - A File Edit View Insert Build Iools	Layout <u>G</u> o F <u>a</u> vorites <u>W</u> indov	× Hep ■ <b>F = ■ ▼ ? A</b> = □ ■ 4	
	AE_REQUEST.ENG (Pane	el)	
	Request Number: Process Frequency: Product: Application: Fields Field	+	
		9 9	
N Ball 人 Fiel Classif Polences 人 Ugents 人 Fee	uts / Viblas /		×
Ready			

Figure 42.7 Cloning the AE\_REQUEST panel



Figure 42.8 Saving the panel with our new name

Let's save the panel using the name MY\_RUN\_CNTL\_AE (figure 42.8).

Our panel is now complete.

**NOTE** When cloning, make sure you don't inadvertently save the object under its original name. This is true for all cloned objects—records, panels, and so on.

Figure 42.9 shows the completed panel named MY\_RUN\_CNTL\_AE. We've made some cosmetic adjustments as well. We moved the process request number into the top frame. We also surrounded the user fields Process Type and Process Name with a group box and labeled it "Processing Parameters." Also, notice the inner scroll bar. This contains the fields for our custom record. We had no choice in this matter. PeopleSoft does not allow you to place multiple records under the same scroll bar. We are going to change the inner scroll bar properties so it is not visible to the user.

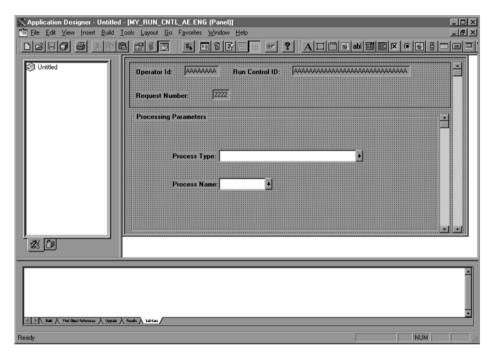


Figure 42.9 Our new Run Control panel is complete

In our previous exercise, we demonstrated the ability to submit multiple process requests. This is one of the features of the AE\_REQUEST panel (which we cloned). For our purposes, we do not want to allow multiple process requests. Deleting process definitions can be considered a dangerous function. Restricting this function to delete one process definition at a time is a wise decision.

Because the only remaining fields within the outer scroll are "Display Only," we cannot insert an additional process request. This is the desired effect. We could have set the outer scroll properties to restrict rows from being inserted and to also make the scroll bar invisible. We'll leave the outer scroll bar properties as they are. Because the inner scroll bar has data entry fields, we have to modify the inner scroll bar properties.

Figure 42.10 displays the inner scroll bar properties we've set. The Occurs count is 1 so only one row may exist as a child of the AE\_REQUEST record. The scroll bar will be invisible. The user will not be able to insert or delete rows within the inner scroll bar (via F7 and F8).

Panel Field Properties	
Label Use	
Scroll Attributes	
Occurs Level: 2	
Occurs <u>C</u> ount: 1	
Field Use Options	
Invisible I Default Width	
No Auto Select 🔲 No Auto Ugdate	
No Row Insert V No Row Delete	
Popup Menu	
Field Help Conte <u>x</u> t Number:	
0 < Auto Assign	
OK Cancel	Figure 42.10
	Inner scroll bar properties

Figure 42.11 shows the panel elements that make up the entire panel. You can access this screen by selecting Layout –Order on the Application Designer menu bar. Make sure your panel entries match those in figure 42.11.

Order Panel			×
Num Lvl Label	Туре	Field	Record
2 1 Request Scroll Bar 3 1 Operator Id 4 1 Run Control ID 5 1 Request Number 6 2 Scroll Bar (Request Option 7 2 Processing Parameters 8 2 Process Type	Frame Scroll Bar Edit Box Edit Box Edit Box Scroll Bar Group Box Edit Box Edit Box	OPRID RUN_CNTL_ID REQUEST_NBR PRCSTYPE PRCSNAME	AE_REQUEST AE_REQUEST AE_REQUEST MY_RUN_CNTL_AE MY_RUN_CNTL_AE MY_RUN_CNTL_AE
OK Can	cel <u>S</u> elect	Move Inselect	<u>D</u> efault

Figure 42.11 The MY\_RUN\_CNTL\_AE panel elements

# 42.4 CREATE A NEW PANEL GROUP

Now, let's create a panel group for our new Application Engine process.

Figure 42.12 shows the panels we've added to our new panel group. Of course, the custom Run Control panel has been added. We've also added the panel AE\_MESSAGE\_LOG. You've seen this panel during our exercises to look at Application Engine messages. We'll attach it to our panel group so we have a convenient means of viewing messages. We've also entered a descriptive label for each panel in the Item Label column.

Navigation:	Go →PeopleToo	ols →Applicatio	n Designer $\rightarrow F$	ile →New -	→PanelGroup
0		T T T	0		

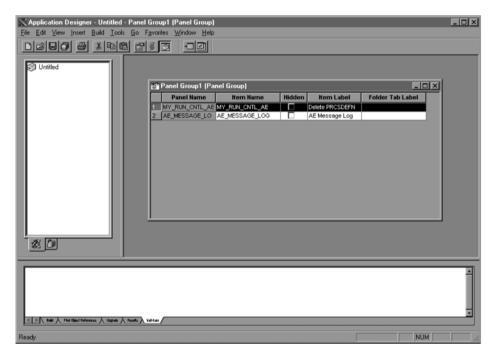


Figure 42.12 Creating the new panel group

**NOTE** Much of the synchronization between the AE\_REQUEST record and our MY\_RUN\_CNTL\_AE record was made in anticipation of adding the AE\_MESSAGE\_LOG panel to our new panel group. It's always a nice touch to give the user access to Application Engine messages from the same panel group.

Before we can save the panel group, we need to enter the panel group properties. You can access the properties by clicking the right mouse button or pressing ALT-ENTER. Figure 42.13 shows the description added for our panel group.

Panel Group Properties	
General Use	
Market	
Delete Process Definitions	
Comments:	
T0-09/04/1999 Panel Group for MY_RUN_CNTL_AE.	
Last Updated	
Date/Time:	
By Operator:	Figure 42.13
OK Cancel	Adding a description to the Panel Group Properties

We've added AE\_REQUEST as the search record in figure 42.14. We've also checked the Add and Update/Display actions. We're ready to save our panel group.

Panel Group Properties	
General Use	
Access Search record:  Add search record:  Add search record:  Detail panet:  3-Tier Execution Location  Panel Group Suid  Application server  Default (application server)  Default (application server)  Actions  Add  Add  Add  Add  Add  Add  Add  A	
OK Cancel	Figure 42.14 Adding a search record to the Panel Group Properties



Figure 42.15 Saving our new panel group

Figure 42.15 shows the panel group as it's being saved. We'll use the name MY\_RUN\_AE.

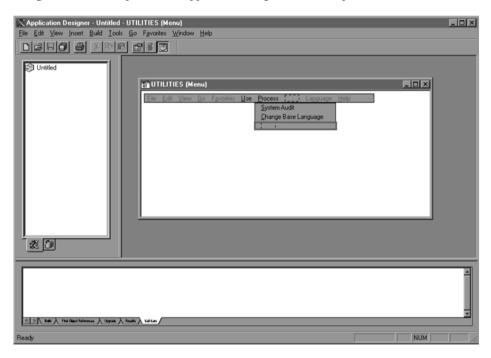
Our next step is to add the panel group to an existing menu. Let's take a step back and review what our process actually does. When the user enters a process type and process name, the application will physically remove all references to it without a trace. If the wrong process type/name is entered it will be deleted! A menu

with limited authorization would be suitable. Only a select few should be running this process. Since this can be considered a PeopleTools utility, it seems logical to add this to the delivered PeopleTools utility menu.

### 42.5 ATTACHING THE PANEL GROUP TO A MENU

Let's attach the panel group to the PeopleTools utility menu.

Figure 42.16 shows the utilities menu. The column labeled "Process" is a perfect place for our new panel group. Let's update the menu item properties for the next available menu item position (the open rectangle).



*Navigation:* Go —PeopleTools —Application Designer —File —Open —Menu —UTILITIES

Figure 42.16 Adding the panel group to the Utilities menu

ATTACHING THE PANEL GROUP TO A MENU

Enter the menu item properties as shown in figure 42.17 using the new panel group MY\_RUN\_AE. Use descriptive text for the menu item label. "Delete Process Definition" is a good choice.

Menu Item P	roperties
- Menu Item	
Name:	MY_RUN_AE
Label:	Delete Process Definition
Туре	
Panel     G     Panel     G	<u>à</u> roup
C People	Code
O S <u>e</u> para	tor
Panel Grou	p
Name:	MY_RUN_AE
Market:	GBL <u>S</u> elect
Search Re	c: AE_REQUEST
C Overrid	e: 🔽
	OK Cancel

Figure 42.17 Entering the menu item properties

Now, click OK and save the utilities menu. Our next step is to assign security to the new menu item.

# 42.6 Assigning Operator Security

We'll assign access to the operator class ALLPANLS. Remember, this process should be limited to a small group of people.

Figure 42.18 shows the security administrator panel. The UTILITIES menu is highlighted for the ALLPANLS operator class. Double-click the UTILITIES Menu to access our new menu item.

*Navigation:* Go  $\rightarrow$ PeopleTools  $\rightarrow$ Security Administrator  $\rightarrow$ File  $\rightarrow$ Open  $\rightarrow$ ALLPANLS  $\rightarrow$  Menu Items  $\rightarrow$ UTILITIES

St. Security A	Administrator - [ALLPANLS (Class of Operators)]
	View Insert Tools Go Favorites Window Help
	<b>6 8 8 8</b> 8
	REPORT_FRENCH_REGULATIONS
	REPORT_PAYROLL_INFO_CANADA
General	REPORT_PAYROLL_INFO_U.S.
	REPORT_TOTAL_COMPENSATION
	REPORT_U.SREGULATIONS
	REPORT_UK_REGULATIONS
Menu Items	SECURITY_ADMINISTRATOR
incrita resints	SQL_SCRIPTER
	TRACK_GLOBAL_ASSIGNMENTS_(CAN)
0	TRACK_GLOBAL_ASSIGNMENTS_(GBL)
	TRACK_GLOBAL_ASSIGNMENTS_(GER)
Signon Times	TRACK_GLOBAL_ASSIGNMENTS_(JPN)
	TRACK_GLOBAL_ASSIGNMENTS_(US)
	TRANSLATE
	TREE_MANAGER
Process	UTILITIES
Groups	W3_EMPLOYEE
I	W3_GENERAL_USER
1 1	W3_MANAGER
	WORKFLOW_ADMINISTRATOR
Ready	

Figure 42.18 Assigning security to the ALLPANLS operator class

You can see our new menu item in figure 42.19. To assign security access to the ALLPANLS class, simply click on the associated MY\_RUN\_AE items. These are the last four items that appear. Once all four items are highlighted, click on the OK button.

elect Menu		_	×
Menu Name: Bar Name	UTILITIES Item Name	Actions/Panels	DispOnly
	OPRALIASVALUE	Update/Display PSOPRALIASTYPE Update/Display PSOPRALIAS	No
PROCESS	SYSTEM_AUDIT RUN_TLSLANG1	Add Update/Display SYSAUDIT_DEFN Add Update/Display RUN TLSLANG1	No
	MY RUN AE	Add Update/Display MY_RUN_CNTL_AE AE_MESSAGE_LOG	No No
<u>S</u> elect All	Change Display-Only	OK	Cancel

Figure 42.19 Our new menu item (MY\_RUN\_AE) as it appears in the menu

Assigning operator security

Now, save the new operator class settings (File  $\rightarrow$ Save). Let's sign off PeopleSoft and log back on so that our new security goes into effect.

# 42.7 TESTING THE NEW PANEL

Let's test the modifications we've made. We haven't actually created the Application Engine program yet, but we can see if our Run Control panel is behaving correctly.

We can immediately tell that our operator security changes were successful: the panel does appear in the menu. Using a test Run Control ID (MYTEST), we can successfully select any process type and process name from the drop-down lists (figure 42.20). An important test would be saving the Run Control record. In our case, the record was saved without a problem. Part of the reason for our successful result was due to the PeopleCode we put in place to populate the AE\_PRODUCT and AE\_APPL\_ID fields in the AE\_REQUEST record. This alleviated the required field constraint found on the AE\_REQUEST record.

Utilities - Process - Delete Process Definition File Edit View Go Favorites Use Process Help	€ + ✓ ৵ /	
Delete PRCSDEFN AE Message Log		
Operator Id: PS Run Control ID: MYTEST		*
Request Number: 1		
Processing Parameters Process Type: SQR Report Process Name: ABS001		
	Delete PRCSDEFN	Add //

Figure 42.20 Our new menu item (MY\_RUN\_AE) as it appears in the menu

We can verify that the rows in both tables are being saved correctly using the database's query tool. Figure 42.21 shows the results of queries made against both tables.

CHAPTER 42 USING RUN CONTROLS—PART A

Notice the second query has the product and application ID populated correctly. This was assigned by the PeopleCode we placed in the MY\_RUN\_CNTL\_AE record. Everything seems to be working as planned.

Untit	led - SQLTalk for Windows					
ile <u>E</u> di	t <u>S</u> ession <u>V</u> iew <u>O</u> ptions <u>H</u> e	яþ				
0 / 2		D % & 0 6 6 5				
elect *	from ps_my_run_cntl_ae wh prid, run_cntl_id, request_nl	ere run_cntl_id = 'MYTEST	'; from ps. ae. request w	where run, onthi	d = 'MYTEST'·	
	pha, ran_cha_la, request_n	n, ac_product, ac_appi_id	from ps_ae_request #	mere run_cha_r	u - MITEST,	
						_
PRID	RUN_CNTL_ID	REQUEST_NBR	PRCSTYPE		PRCSNAME	_
	RUN_CNTL_ID		PRCSTYPE SQR Report		PRC SNAME AB \$ 001.	
PRID 5 ROW 5						
5	MYTEST					
5	MYTEST	1		21_10		
ROW S	MYTEST SELECTED	REQUEST_NBR	SQR Report	_		
ROW S PRID	MYTEST SELECTED RUN_CNTL_ID	REQUEST_NBR	SQR Report <u>AE_PRODUCT</u> <u>AE_APP</u>	_		
ROW S PRID	MYTEST SELECTED RUN_CNTL_ID NYTEST	REQUEST_NBR	SQR Report <u>AE_PRODUCT</u> <u>AE_APP</u>	_		

Figure 42.21 Looking at the resulting rows using SQL\*Talk

Our next step is to create a process definition for our application.

### 42.8 CREATING OUR PROCESS DEFINITION

We'll now create the process definition for our application. The process type is Application Engine. The name of our application is MYPRCSDL.

We've added the process type and process name (figure 42.22). Let's enter the process definition information into Process Scheduler.

*Navigation:* Go  $\rightarrow$ PeopleTools  $\rightarrow$ Process Scheduler  $\rightarrow$ Use  $\rightarrow$ Process Definitions  $\rightarrow$  Process Definitions  $\rightarrow$ Add

Add Process Definitions	X	
Process Type:     Application Engine       Process Name:     MYPRCSDL	OK Cancel	Figure 42.22 Adding our process definition

Figure 42.23 shows the process definition information we need to add. We've added our new MY\_RUN\_AE panel group to the definition screen along with additional items such as descriptive text and process security groups. The process class of Application Engine programs is COBOL SQL.

Eile Edit View G	uler - Use - Process D o Favorites Use Pro Favorites Definition Option	cess Help	<u>@</u> (-) <b>+</b> ✓	∕ ✔ ℤ	_ 🗆 🗙
Process Type: Process Name:	Application Engine MYPRCSDL				
Description:	Delete Process Defin	nition			
Process Class:	COBOL SQL		+	Log client requ	est
Server Name:	+			SQR Runtime	
Priority:	Medium 💌				
Run Location:	Both 💌		e:		+
Long Description:	TD-09/04/1999 Applicat	ion Engine Utility to de	elete process definitions.		14. 14
Panel Groups			ess Security Groups		. 🗉
MY_RUN_AE	+		RALL		•
		. <b>₽</b>	YALL		•
			Process	Definitions	Add //

Figure 42.23 Adding process definition details

Our process definition is complete.

#### 42.8.1 Create a DUMMY process definition for testing

While we're in Process Scheduler, let's create a dummy process definition that we'll use to test our application. We don't want to delete any existing process definitions. Let's add the dummy definition. We'll load random information since it's going to be deleted by our process.

Figure 42.24 shows a sample definition of a DUMMY process. We added a random panel group and process security groups. We'd like to see if our new application will delete process definition entries from a variety of tables. This panel will now contain DUMMY entries for the tables PRCSDEFN, PRCSDEFNPNL, and PRCSDEFNGRP.

We've also added some random panel transfer information for our DUMMY process definition. This will create a DUMMY entry in the table PRCSDEFNXFER.

Now, all that's left to do is create the actual Application Engine program!

	uler - Use - Process I o Favorites Use Pro Process Definition Option	ocess Help	nsfers		<b>~</b>	
Process Type: Process Name:	Crw Online DUMMY					
Description:	DUMMY Entry		_			
Process Class:	Crystal		+		Log client requ	lest
	+				SQR Runtime	
Priority:	Medium 🔹					
Run Location:	Both 💌		e Name:			+
Long Description:	TD-09/04/1999. DUMN	fY Process De	finition to test MYP	RCSDL A/E Appl	ication.	*
Panel Groups			Process Secu	urity Groups		
TEST_TABLE	*		HRALL			•
			PAYALL			•
				Process [	Definitions	Add //.

Figure 42.24 Adding a dummy process definition to test our application

Process Scheduler - U File Edit View Go Favo Process Definitions Process Process Type: Crw 0 Process Name: DUM	nites <u>U</u> se <u>P</u>	rocess Help		@ <b>(⊅</b>  ⇒)	<u>+</u>   <b>√</b>   <b>√</b>  ∄	
Panel Transfer Code: Menu Name: Menu Bar Name: Menu Item: Panel Name:	ADMINISTE INQUIRE 26 + EMPLOYEE	EMPLOYEE	E_DATA_SUN	IMARY	+	
Menu Action:	Update				Panel Transfers	Correct

Figure 42.25 Adding a DUMMY process definition (Transfers)

### **KEY POINTS**

- 1 The development life cycle for Application Engine programs is identical to that of SQR or COBOL development. The only difference is the Application Engine program itself.
- 2 When creating a Run Control record for Application Engine programs, use the AE\_REQUEST record as a shell. Application Engine always uses the AE\_REQUEST record so it's a good idea to integrate your new Run Control record with it. Use RowInit PeopleCode to assign your program name to the AE\_PRODUCT and AE\_APPL\_ID fields in the AE\_REQUEST record.
- **3** Also clone the AE\_REQUEST panel when creating a new Run Control panel. The AE\_REQUEST record will be the parent to your new Run Control record.
- **4** When creating the new panel group, add the AE\_MESSAGE\_LOG panel after your new Run Control panel This gives the user easy access to the message log entries for the completed run. The search record for your new panel group will be AE\_REQUEST.



Using Run Controls— part B

43.1 Create the Application Engine program 90843.2 Testing the completed application 933

The Run Control panel for our new utility program is complete. We can access the new panel on the menu and even enter Run Control parameters. That's as far as we can go at the moment. If we click on the Traffic Light to initiate the process, an error will occur. That's because we haven't created the Application Engine program yet. This chapter will concentrate on the development of the Application Engine program. Once created, we can initiate the process through the Run Control panel. Some careful planning must be made to structure our program properly. Once complete, we can begin using our new tool to delete obsolete process definitions.

#### 43.1 **CREATE THE APPLICATION ENGINE PROGRAM**

We're ready to begin developing our Application Engine program. It may be helpful to give a brief overview of our program structure:

MAIN.STEP1	Obtain Run Control Parameters
MAIN.STEP2	Display Run Control Parms on Message Log
MAIN.STEP3	Fetch Table One by One (DO Select)
DYNSECTN.STEP1	Dynamically call PROCESS1 or PROCESS2
PROCESS1.STEP1	Determine Number of Rows
PROCESS1.STEP2	Process if exists (DO When > 0)
DELETE1.STEP1	Delete Process Definition from table
PROCESS1.STEP3	Call Message Routine
MESSAGE.STEP1	Display Message
PROCESS2.STEP1	Determine Number of Rows
PROCESS2.STEP2	Process if exists (DO When > 0)
DELETE2.STEP1	Delete Process Definition from table
PROCESS2.STEP3	Call Message Routine
MESSAGE.STEP1	Display Message

We need to make two key points. Consider the structure of our Application Engine program. The first key point has to do with the MAIN.STEP3 line. This step will select each of the six tables and process them one by one. Early in this chapter, we pointed out that four of the process definition tables are prefixed with "PS\_", and two are not. We'll use two separate processes (PROCESS1 and PROCESS2) to handle both types. They will be called dynamically based on the table being processed. The step DYNSECTN.STEP1 will call either PROCESS1 or PROCESS2.

The second point has to do with the processing steps we have chosen. Some readers may think we have taken the long way in performing our task. This may be true depending on the database you are using. For instance, if you're an Oracle user, you certainly don't need to determine if the row exists before deleting it. This is not true for all databases though. In DB2, you may receive an error if you try to delete a row that doesn't exist. We would then need to add additional error-handling steps. We could also use the DB platform field to code individual routines based on your particular database. This would hardly seem practical for the purposes of this book. Let's move on now.

We'll begin developing our program from the minor routines on up. Since we've established the program hierarchy, we can work backward and not worry about step dependencies we would encounter by going forward.

CHAPTER 43 USING RUN CONTROLS—PART B

#### 43.1.1 Building the MESSAGE section

We'll create the section MESSAGE first.

*Navigation:* Go  $\rightarrow$ PeopleTools  $\rightarrow$ Application Engine  $\rightarrow$ Use  $\rightarrow$ Application Engine  $\rightarrow$  Application  $\rightarrow$ Add

Add Application E	ngine	×	
Product:	PS/AE	OK	
Application:	MYPRCSDL	Cancel	
Section:	MESSAGE		
Database Platform			Figure 43.1 Creating the
			MESSAGE section

First tab over to the application definition (figure 43.2). Fill in the description, cache record, version, and message set number. We'll be using our updated cache record (USER\_AET) and message set from prior exercises. Also, set the trace parameter to SQL. After we test our application, we'll examine the trace file.

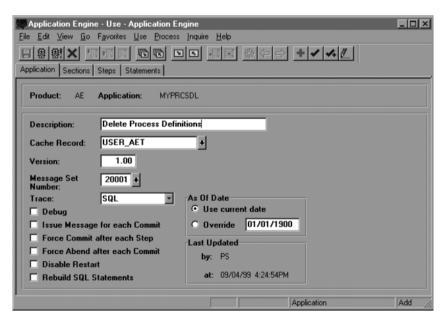


Figure 43.2 Defining our application MYPRCSDL

Fill in the description of the MESSAGE section (figure 43.3). This section simply writes a message to the message log.

	Favorites Use E	trocess Inqui		MESSAGE	Platform:	
Effective Date: Description: Commit after ea Type Preparation O Critical Datab Actions New S	nly ase Updates		Active			•
				Sec	tions	Add //

Figure 43.3 Defining the MESSAGE section

The only parameter you need to fill in for the first (and only) step of the MES-SAGE section is the step name. We call it STEP1 (figure 43.4).

Figure 43.5 shows the message statement we use. This is similar to prior exercises. We pass the record name (RECNAME) and number of rows (COUNTER) to the message log.

We're done with the MESSAGE section.

Application Engine - Use - Application Engine		<u>_                                    </u>
Elle       Edit       View       Go       Favorites       Use       Process       Inquin         Image: State and the st		
Product: AE Application: MYPRCSDL	Section: MESSAGE DB Pltfrm:	
Effective Date: 09/04/1999	Cache Record: USER_AET	-
Step:     STEP1     Status:     Active       Image: SQL Statement     Edit	Step Position	
Mass Change     D0 Section     Edit     COBOL Program	DO Edit DO Select	
Commit © Default Action © After this Step © Later © Suppress Error	D0 Method     D0 When       © Select and Fetch     D0 Until       © Re-Select     D0 While	
	Steps	Add //

Figure 43.4 Defining STEP1 of the MESSAGE section

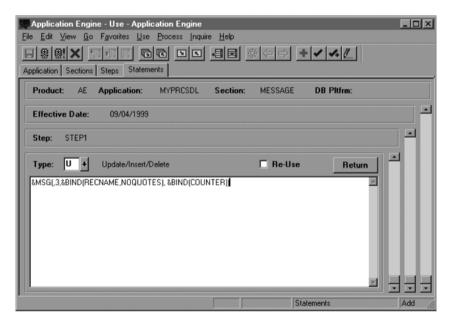


Figure 43.5 Adding our message statement

### 43.1.2 Building the DELETE1 section

We now add the DELETE1 section (figure 43.6), which performs the Delete against the process definition tables that have the standard 'PS\_' prefix.

Add Application	Engine		×	
Product:	PS/AE	•	ОК	
Application:	MYPRCSDL		Cancel	
Section:	DELETE1			Figure 42.6
Database Platfor	m:	•		Figure 43.6 Adding the DELETE
				section

The only thing we need to add is the description. 'Delete Table with PS\_' is a fairly accurate description (figure 43.7).

Application Engine - Use - Application Engine         File Edit View Go Favorites Use Process Inquire Help         Image: State	
Product: AE Application: MYPRCSDL Section: DELETE1 Platform:	
Effective Date:       09/04/1999       Status:       Active       ▼         Description:       Delete Table with PS       •       •       •         Commit after each step       Type       •       •       •       •         O Preparation Only       •       •       •       •       •       •         Actions       •       Actions       • </th <th>×</th>	×
Sections	Correct //

Figure 43.7 Defining the DELETE1 section

We call this first (and only) step of the DELETE1 section STEP1 (figure 43.8). We've added the SQL statement to Delete a row from the table specified by the RECNAME cache field (figure 43.9). Notice the table name is in the same dynamic format used in prior exercises. The DELETE1 section will be part of the process that handles records with the PS\_ prefix. The DELETE1 section is complete.

<u>File Edit View Go Favo</u>		ie <u>H</u> elp ≝≣ ŵ≎়⊳	+ <b>/</b> /	
Product: AE Appl	ication: MYPRCSDL	Section: DELETE1	DB Pltfrm:	
Effective Date: 09	/04/1999	Cache R	ecord: USER_AET	-
Step: STEP1 © SQL Statement © Mass Change	Status: Active	• 1 St	ep Position	
O DO Section	Edit		DO Edit Exis	ts
C COBOL Program			DO Select	
Commit C Default Action After this Step Later	If Unsuccessful Abort Application Ignore Error Suppress Error	DO Method © Select and Fetch © Re-Select	D0 When	
			Steps	Correct //

Figure 43.8 Defining STEP1 of the DELETE1 section

ire <u>H</u> elp	-OX
Section: DELETE1 DB Pltfrm:	
Cache Record: USER_AET	-
T Step Position	
x t ¥ 3	
+ DO Edit	
	<u>kists</u>
DU Method	
O Re-Select	
Steps	
	Section: DELETE1 DB Pltfrm: Cache Record: USER_AET

Figure 43.9 Defining STEP1 of the DELETE1 section

CREATE THE APPLICATION ENGINE PROGRAM

#### 43.1.3 Building the DELETE2 section

We add the DELETE2 section (figure 43.10). This section performs the Delete against the process definition tables that DO NOT utilize the standard PS\_ prefix.

Add Application Er	igine	×	
Product:	PS/AE	OK	
Application:	MYPRCSDL	Cancel	
Section:	DELETE2		Figure 43.10
Database Platform			Adding the DELETE2
		5	section

Once again, the only thing we need to add is the description. 'Delete Table without  $PS_{'}$  is perfect for our section (figure 43.11).

Application Engine - Use - Application Engine File Edit View Go Favorites Use Process Inquire Help B B X F F F F F F F F F F F F F F F F F	
Product: AE Application: MYPRCSDL Section: DELETE2 Platform:	
Effective Date: 09/04/1999 Status: Active  Description: Delete Table without PS_ Commit after each step Type Preparation Only Critical Database Updates Actions New Save As Rename Delete	
Sections	Add //

Figure 43.11 Defining the DELETE2 section

**TIP** Existing sections can be "cloned" using the Save As button on the Section Definition panel. We can then modify the new section as needed. In the case of the DELETE1 and DELETE2 sections, the modifications are minimal. For purposes of these exercises, we'll create each section manually.

Let's call this first (and only) step of the DELETE2 section STEP1 (figure 43.12).

Application Engine - Use - Application Engine		_OX
Elle     Edit     View     Go     Favorites     Use     Process     Inquir       Image: State and State		
Product: AE Application: MYPRCSDL	Section: DELETE2 DB Pltfrm:	
Effective Date: 09/04/1999	Cache Record: USER_AET	·
Step:         STEP1         Status:         Active           © SQL Statement         Edit	T Step Position	-
C Mass Change	+ D0 Edit	
C DO Section Edit	D0 Select	Exists
Commit Default Action After this Step Later Commit If Unsuccessful Abort Application Ignore Error Suppress Error	D0 Method         D0 When           © Select and Fetch         D0 Until           © Re-Select         D0 While	
	Steps	

Figure 43.12 Defining STEP1 of the DELETE2 section

The delete statement in figure 43.13 is almost identical to the one in the DELETE1 section. The only difference is the absence of the PS\_ prefix.

The DELETE2 section is complete.

Product:       AE       Application:       MYPRCSDL       Section:       DELETE2       DB       Pltfrm:         Effective Date:       09/04/1999       Image: Step:       Step:       STEP1       Image: Step:       Image: Step: Step:       Image: Step: Step:       Image: Step: Step: Step:       Image: Step: St	Application Engine - Use - Application Engine File Edit View Go Favorites Use Process Inquir Process Inquir Application Sections Steps Statements		✓ ✓ /	
Type:     U     Update/Insert/Delete     Re-Use       DELETE     FROM &BIND(RECNAME.NOQUOTES.STATIC)       WHERE PRCSTYPE = &BIND(PRCSTYPE)		Section: DELETE2	DB Pltfrm:	
FROM &BIND(RECNAME,NOQUOTES,STATIC) WHERE PRCSTYPE = &BIND(PRCSTYPE)		E Re-Use	Return	-
	FROM &BIND(RECNAME,NOQUOTES,STATIC) WHERE PRCSTYPE = &BIND(PRCSTYPE)			Ĩ

Figure 43.13 The Delete statement for DELETE2.STEP1

#### 43.1.4 Building the PROCESS1 section

Next, let's develop the PROCESS1 section. This section will handle all process definition tables that require the PS\_ prefixed to the RECNAME. Add the PROCESS1 section now (figure 43.14).

ļ	Add Application En	igine		X	
	Product:	PS/AE	•	OK	
	Application:	MYPRCSDL		Cancel	
	Section:	PROCESS1			Figure 43.14
	Database Platform:		•		Adding the PROCESS1
					section

Fill in the description for the PROCESS1 section. 'Process Table with  $PS_{'}$  is the description we'll use for our section (figure 43.15).

Our first step in this section determines the number of rows in the particular process definition table, which is stored in the RECNAME cache field. The only parameter we need to enter in the Step Definition panel is the name of our step. We'll call it STEP1 (figure 43.16).

Application Engin File Edit View Go P P P P P P P P P P P P P P P P P P P	Favorites Use	Process Inqu	ire <u>H</u> elp	PROCESS1	Platform:	
Effective Date: Description: Commit after of Type Preparation Critical Data Actions New	each step Only	Status: oles with PS_	Active			×
				Sec	tions	Add //

Figure 43.15 Defining the PROCESS1 section

Application Engine - Use - Application Engine File Edit View Go Favorites Use Process Inquir		_ 🗆 🗙
Application Sections Steps Statements		,
Product: AE Application: MYPRCSDL	Section: PROCESS1 DB Pltfrm:	
Effective Date: 09/04/1999	Cache Record: USER_AET	-
Step: STEP1 Status: Active	Step Position	-
• SQL Statement Edit		
O Mass Change	+	
O DO Section Edit	DO Edit Exists	
C COBOL Program	D0 Select	
Commit If Unsuccessful	DO Method DO When	
	© Select and Fetch DO Until	
C Later C Suppress Error	D0 While	
	Change	
Commit Constitution Constitutio	D0 Edit Exists D0 Select © Select and Fetch © Re-Select	Add

Figure 43.16 Adding STEP1 to the PROCESS1 section

We add a simple Select statement to retrieve the number of rows in the table (specified in RECNAME) and populate the COUNTER cache field.

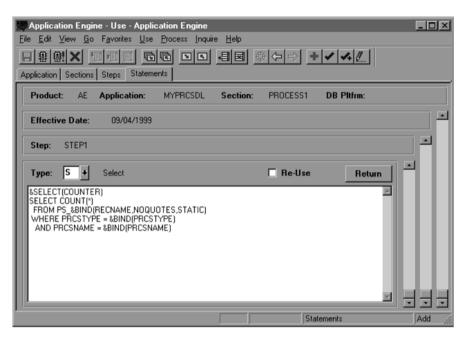


Figure 43.17 Adding the Select statement to PROCESS1.STEP1

Let's add another step to the PROCESS1 section. Click on the Steps Folder tab to return to the Step Definition panel. Place the cursor in the Step field and press the F7 key to insert a new row. Our next step will be named STEP2 (figure 43.18). Click on the DO section radio button then click on the corresponding edit button. Add the DELETE1 section when the DO section dialog box appears. You can see the DELETE1 section name next to the DO section edit button when you return. Our next step is to populate the statement panel with a DO When statement. The DELETE1 section is performed only if there are rows in the table containing the process definition from the Run Control record. Let's add the DO When statement now.

This is the same statement we've used in previous exercises. If the COUNTER cache field contains a value greater than zero, a True condition is returned, and the section DELETE1 is performed. If the COUNTER cache field contains a value of zero, a False condition is returned, and the DELETE1 section is not performed.

CHAPTER 43 USING RUN CONTROLS—PART B

💭 Application Engine - Use - Applic	ation Engine			_ 🗆 🗵
<u>File Edit View Go</u> F <u>a</u> vorites <u>U</u> se				
		\$ ← → + ✓	✓+ /	
Application Sections Steps Stateme	nts			
Product: AE Application:	MYPRCSDL Section	: PROCESSI DB	Pltfrm:	
Effective Date: 09/04/1999		Cache Record:	USER_AET	-
Step: STEP2 Status:	Active	2 Step Pos	ition	
SQL Statement Edit	]		± Ŧ エ	
C Mass Change	+			
DO Section     Edit	DELETE1	DO E	dit Exists	
C COBOL Program		D	D Select	
Commit If Unsuce		thod	0 When	
			0 Until 🗆	
C After this Step C Ignor C Later C Supp	e Error C Re ress Error	-Select D	0 While	
S Later	1622 EII01			
		Steps		Add //

Figure 43.18 Adding STEP2 to the PROCESS1 section

	_	ne - Use - Applic	_				_ 🗆 🗵
		Favorites Use			a de la	<ul> <li>✓</li> <li>✓</li> </ul>	
Application S							
Product:	AE	Application:	MYPRCSDL	Section:	PROCESS1	DB Pltfrm:	
Effective	Date:	09/04/1999					-
Step: 9	STEP2						-
Туре: Н	H +	Do When			🗖 Re-Use	Return	
&SELECT(A SELECT X FROM PS	_	DE)				(A)	
		UNTER) > 0					
						<b>•</b>	
,					Sta	tements	Add //

Figure 43.19 Adding the DO When statement to PROCESS1.STEP2

CREATE THE APPLICATION ENGINE PROGRAM

Use the F7 key again to insert a new step in the PROCESS1 section. Let's call it STEP3 (figure 43.20). Its function is to call the MESSAGE section we've created. Use the DO section radio button and edit box to set the section to MESSAGE. The MESSAGE section simply writes a message log entry containing the record name and number of rows processed.

The PROCESS1 section is complete.

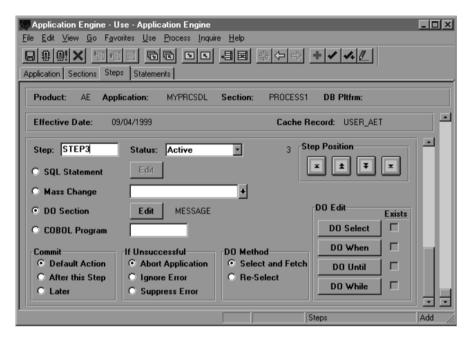


Figure 43.20 Adding STEP3 to the PROCESS1 section

#### 43.1.5 Building the PROCESS2 section

Next, let's develop the PROCESS2 section. This section handles all process definition tables that do not require the PS\_ prefix to the RECNAME. The PROCESS2 section is almost identical to the PROCESS1 section. Add the PROCESS2 section now (figure 43.21).

Add Application E	ngine		X	
Product:	PS/AE	¥	OK	
Application:	MYPRCSDL		Cancel	
Section:	PROCESS2			Figure 43.21
Database Platform	к: [	•		Adding the PROCESS2
			2	section

Fill in the description for the PROCESS2 section. 'Process Table without PS\_' is the description we'll use for our section (figure 43.22).

		ss <u>I</u> nquire <u>H</u> elp	<b>₽</b> ₽	✓ ⊀ ℤ	<u>-</u> 0×
Product: AE	Application: MYF	PRCSDL Section:	PROCESS2	Platform:	
Effective Date: Description: Commit after Type O Preparation Critical Data Actions New	Process Tables wi				×
,			Sec	tions	Add //

Figure 43.22 Defining the PROCESS2 Section

**TIP** You can try cloning the PROCESS1 section to produce the new section PROCESS2 using the 'Save As' button. Make the alterations to the statements as you would if you had created the PROCESS2 section manually.

Our first step in this section determines the number of rows in the particular process definition table stored in the RECNAME cache field. The only parameter we need to enter in the Step Definition panel is the name of our step. We'll call it STEP1 (figure 43.23).

Next, we add a simple Select statement to retrieve the number of rows in the table (specified in RECNAME) and populate the COUNTER cache field. It is nearly identical to the Select statement found in STEP1 of the PROCESS1 section, the difference being the absence of the PS\_ prefix preceding the table name.

Application Engine - Use - Applic	_			_ 🗆 🗙
Eile     Edit     View     Go     Favorites     Use       Image: Section Section     Steps     Statement		©( <b>↓</b> ) <b>+</b> ✓	*	
Product: AE Application:	MYPRCSDL Section:	PROCESS2 DB	Pltfrm:	
Effective Date: 09/04/1999		Cache Record:	USER_AET	-
Step: STEP1 Status:	Active	1 Step Pos	ition	P
SQL Statement Edit			± ∓ ±	
O Mass Change	+			
O DO Section Edit		DO E	dit Exists	
C COBOL Program		DI	D Select 🗖	
Commit If Unsuce	essful DO Meth	od D	0 When	
			O Until	
C After this Step C Ignor C Later C Supp	e Error C Re-S		0 While	
		Steps		Add //

Figure 43.23 Adding STEP1 to the PROCESS2 section

Application Engine - Use - Application Engine		
<u>File Edit View Go Favorites Use Process Inquire</u>	Help	
	-∃ 図  @ (→ →) + ✔ ↓ ℓ	
Application Sections Steps Statements		
Product: AE Application: MYPRCSDL	Section: PROCESS2 DB Pltfrm:	
Effective Date: 09/04/1999		<u> </u>
Step: STEP1		-
Type: S + Select	Re-Use Return	
&SELECT(COUNTER) SELECT COUNT(*) FROM &BIND(RECNAME.NOQUOTES.STATIC) WHERE PRCSTYPE = &BIND(PRCSTYPE) AND PRCSNAME = &BIND(PRCSNAME)		
	Statements	Add

Figure 43.24 Adding the Select statement to PROCESS2.STEP1

Let's add another step to the PROCESS2 section. Click on the Step Folder tab to return to the Step Definition panel. Place the cursor in the Step field and press the F7 key to insert a new row. Our next step is named STEP2 (figure 43.25). Click on the DO section radio button, then on the corresponding edit button. Add the DELETE2 section when the DO section dialog box appears. You can see the DELETE2 section name next to the DO section edit button when you return. Our next step is to populate the statement panel with a DO When statement. The DELETE2 section is performed only if there are rows in the table containing the process definition from the Run Control record. Let's add the DO When statement now.

Application Engine - Use - Application       File     Edit     View     Go     Favorites     Use       Image: Second Sec	Process Inquire Help	◎	
Product: AE Application:	MYPRCSDL Section:	PROCESS2 DB Pltfn	 
	Active	2 Step Position	
Mass Change     DO Section     Edit	• DELETE2	DO Edit	Exists
C COBOL Program		d DO Se DO W t and Fetch DO U	hen
C After this Step C Ignore		elect DO W	hile
		Steps	Add

Figure 43.25 Adding STEP2 to the PROCESS2 section

Our DO When statement (figure 43.26) is the same as before. If the COUNTER cache field contains a value greater than zero, a True condition is returned, and the section DELETE1 is performed. IF the COUNTER cache field contains a value of zero, a False condition is returned, and the DELETE2 section is not performed.

	ew <u>G</u> o X <u>म</u>				\$ <b>\$</b>		
Product:	AE	Application:	MYPRCSDL	Section:	PROCESS2	DB Pltfrm:	
Effective	Date:	09/04/1999					-
Step: S	TEP2						
Туре: Н	<del>i</del> +	Do When			🗖 Re-Use	Return	
&SELECT X SELECT X FROM PSI WHERE &E	LOCK	DE) UNTER) > 0					
,					Sta	tements	Add

Figure 43.26 Adding the DO When statement to PROCESS2.STEP2

Use the F7 key again to insert a new step in the PROCESS2 section. Let's call it STEP3 (figure 43.27). Its function is to call the MESSAGE section we've created. Use the DO section radio button and edit box to set the section to MESSAGE. As stated in the creation of STEP3 in the PROCESS1 section, the MESSAGE section simply writes a message log entry containing the record name and number of rows processed.

The PROCESS2 section is complete.

Application Engine - Use - Application	-			_ 🗆 🗵
Elle     Edit     Yiew     Go     Favorites     Use       Image: Section State     Image: Section State     State     State     State		© (∽   → (<	<b>⊀</b> ₊ <u>∥</u>	
Product: AE Application:	MYPRCSDL Section:	PROCESS2 DB I	Pltfrm:	
Effective Date: 09/04/1999		Cache Record:	USER_AET	-
Step: STEP3 Status:	Active 🔹	3 Step Posi		P
C SQL Statement Edit			± ∓ ⊥	
C Mass Change	+			
DO Section Edit	MESSAGE	DO Ed	lit Exists	
C COBOL Program		DO	Select	
Commit If Unsucc	essful DO Meth	DC	When 🗆	lat t
			O Until 🗖	
C After this Step C Ignore	Error C Re-S		) While	
		Steps		Add //

Figure 43.27 Adding STEP3 to the PROCESS2 section

#### 43.1.6 Building the DYNSECTN section

Add a new section called DYNSECTN (figure 43.28). The purpose of this routine is to dynamically call either PROCESS1 or PROCESS2.

Add Application	Engine		×	
Product:	PS/AE		OK	
Application:	MYPRCSDL		Cancel	
Section:	DYNSECTN			Figure 43.28
Database Platfo	rm:	•		Adding the DYNSECTN
				section

First, add the description for the DYNSECTN section (figure 43.29).

Application Engine           Ele         Edit         View         Go           Image: Section S	Favorites <u>U</u> se	Process Inqui		• <b>(</b> -) •	✓ 4. L	
Product: AE	Application:	MYPRCSDL	Section:	DYNSECTN	Platform:	
Effective Date: Description: Commit after of Type Preparation Critical Data Actions New	each step Only abase Updates	Status: amic Section	Active Delete			×
				Sect	tions	Add //

Figure 43.29 Defining the DYNSECTN section

On the Step Definition panel fill in the step name with STEP1 (figure 43.30).

Application Engine - Use - Applicati File Edt View Go Favorites Use Pri	rocess Inquire Help		
Product: AE Application: M Effective Date: 09/04/1999	AYPRCSDL Section: DYNSECT Cache	IN <b>DB Pitfrm:</b> Record: USER_AET	
Step:     STEP1     Status:     A       C     SQL Statement     Edit       C     Mass Change	Active Y 1	Step Position	-
DD Section     Edit     COBOL Program		DO Edit Exists DO Select DO When	
Commit If Unsucces Default Action Abort Ap After this Step Ignore E Later Suppres	pplication © Select and Fetc Error © Re-Select		
		Steps	Add //

Figure 43.30 Adding STEP1 to the DYNSECTN section

CHAPTER 43 USING RUN CONTROLS—PART B

Next, click on the DO section radio button and press the corresponding edit button. When the DO section dialog box appears, click on "Dynamic Section." Notice the literal DYNAMIC (in brackets) next to the DO section edit button (discussed in chapter 41). The symbolic parameter &SECTION is placed in the AE\_DO\_SECTION column in the AE\_STEP\_TBL. This is the table that's populated by your Step Definition panel entries. When the step is executed, the value of the AE\_SECTION cache field will be used in place of the &SECTION symbolic. Our next and final section, MAIN, populates the AE\_SECTION cache field with the value PROCESS1 or PROCESS2. The DYNSECTN section then performs PROCESS1 or PROCESS2.

The DYNSECTN section is complete.

## 43.1.7 Building the MAIN section

As we've discussed, all Application Engine programs must begin with a section called MAIN. Let's add the MAIN section now (figure 43.31).

	×	[	n Engine	Add Application Er
		OK	PS/AE	Product:
Section: MAIN	el	Cancel	MYPRCSDL	Application:
			MAIN	Section:
Database Platform:			form:	Database Platform

Figure 43.31 Adding the MAIN section

The Section Definition panel is displayed in figure 43.32. We added a simple description of the MAIN section: Delete Process Definitions.

We'll call the first step of the MAIN section STEP1 (figure 43.33). The purpose of this step is to obtain the processing parameters from our Run Control record.

Application Engine - Use - Application Engine File Edit View Go Favorites Use Process Inquire Help POR CONSTRUCTION Sections Steps Statements	X
Product: AE Application: MYPRCSDL Section: MAIN	Platform:
Effective Date: 09/04/1999 Status: Active   Description: Delete Process Definitions  Commit after each step  Type  Preparation Only  Critical Database Updates  Actions  New Save As Rename Delete	•
Sect	ons Add //

Figure 43.32 Defining the MAIN section

	■E®©≎₽ <u>+√∢ℓ</u>	_ 🗆 🗙
Application Sections Steps Statements   Product: AE Application: MYPRCSDL  Effective Date: 09/04/1999	Section: MAIN DB Pltfrm: Cache Record: USER_AET	
Step:         STEP1         Status:         Active           ©         SQL Statement         Edit	Step Position	
Mass Change     DD Section     Edit     COBOL Program	DO Edit DO Select	
Commit If Unsuccessful Default Action Abort Application After this Step Ignore Error Later Suppress Error	D0 Method     D0 When     C       © Select and Fetch     D0 Until     C       © Re-Select     D0 While     C	
	Steps	

Figure 43.33 Adding STEP1 To the MAIN section

Figure 43.34 shows the Select statement for MAIN.STEP1. To retrieve our run parameters, we need to join our Run Control record (MY\_RUN\_CNTL\_AE) to the Application Engine Run Control record called AE\_RUN\_CONTROL. The AE\_RUN\_CONTROL record discussed earlier in the book is used by Application Engine to hold run information such as PROCESS\_INSTANCE, OPRID, RUN\_CNTL\_ID, and REQUEST\_NBR. It also stores information about each run such as the last step committed, used when restarting an Application Engine program that may have terminated due to errors.

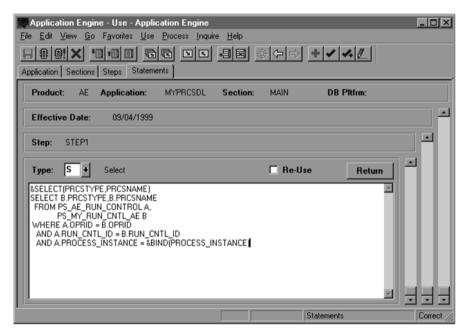


Figure 43.34 Select statement to retrieve our Run Control parameters

The records MY\_RUN\_CNTL\_AE and AE\_RUN\_CONTROL are joined by the OPRID and RUN\_CNTL\_ID fields. The AE\_RUN\_CONTROL record is selected for the appropriate PROCESS\_INSTANCE assigned by the Process Scheduler. The &SELECT statement stores the PRCSTYPE and PRCSNAME Run Control parameters in our cache record. We now have access to the process parameters entered on the Run Control panel. We'll display them on the message log in our next step.

Return to the Step Definition panel by clicking the Steps Folder tab. Place the cursor in the step field and press F7 to insert a new row. We'll call the new step STEP2 (figure 43.35). Now, click on the Statements Folder tab.

We write a simple message to the message log displaying the process type (PRCSTYPE) and process name (PRCSNAME) taken from the Run Control record. You

can see the &MSG syntax in figure 43.36. Remember, the &MSG function only works with a statement Type of "U" (Update/Insert/Delete).

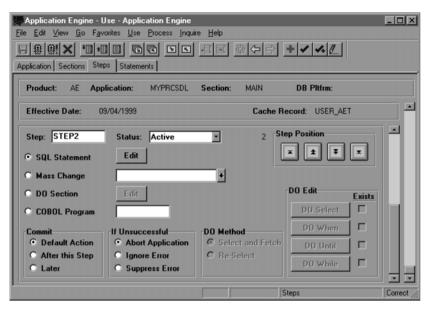


Figure 43.35 Adding STEP2 to the MAIN section

<u>File Edit View Go</u>		-	© <b>( )</b> ●	∕ .≁.∥	
Product: AE	Application: M	YPRCSDL Section	: MAIN	DB Pltfrm:	
Effective Date:	09/04/1999				-
Step: STEP2					-
Type: U +	Update/Insert/Delete		🗖 Re-Use	Return	-
&MSG(,2,&BIND(PR	CSTYPE,NOQUOTES)	&BIND(PRCSNAME,NC	iquotes))		
			Staten	nents	

Figure 43.36 Run Control parameters are written to the message log

CHAPTER 43 USING RUN CONTROLS—PART B

On the Step Definition panel, press F7 to insert another step. We'll call it STEP3 (figure 43.37). Click on the DO section radio button and the corresponding edit button. When the DO section dialog box appears, enter the section DYNSECTN. This is the section that handles the dynamic call of either PROCESS1 or PROCESS2. Let's code the DO Select statement for STEP3 in our MAIN section.

Application Engine - Use - Application Engine         File       Edit       View       Go       Favorites       Use       Process       Inquire       Help         Image: State State Statements       Image: Statements	<u> ×</u>
Product: AE Application: MYPRCSDL Section: MAIN DB Pltfrm:	
Effective Date: 09/04/1999 Cache Record: USER_AET	-
Step:     STEP3     Status:     Active     3     Step Position       C     SQL Statement     Edit     Image: Step Position     Image: Step Position       C     Mass Change     Image: Step Position     Image: Step Position	
DO Section     Edit     DYNSECTN     DO Edit     Exists     DO Select     D	
Commit If Unsuccessful D0 Method D0 When D0 Until D0 Unti	
C Suppress Error DO While Steps	Correct

Figure 43.37 Adding STEP3 to the MAIN section

Figure 43.38 shows the DO Select statement that controls the processing of the six Process Definition tables. Let's take a closer look at the statement and describe what's happening:

&SELECT(AE_SECTION, RECNAME)		Assign cache field values
SELECT 'PROCESS1', RECNAME		Select process definition tables that require the
FROM PSRECDEFN		PS_ prefix. Each row selected returns two columns
WHERE RECNAME = 'PRCSDEFN'		- 'PROCESS1' and RECNAME. 'PROCESS1'
OR RECNAME = 'PRCSDEFNGRP'		populates the AE_SECTION cache field.
OR RECNAME = 'PRCSDEFNPNL'		
OR RECNAME = 'PRCSDEFNXFER	1	
UNION	4	Using a UNION, select process definition tables that do not use the PS_ prefix. The
SELECT 'PROCESS2', RECNAME		rows returned will have the columns
FROM PSRECDEFN		'PROCESS2' and RECNAME. 'PROCESS2' will
		be stored in the AE SECTION cache field.
WHERE RECNAME = 'PSPRCSRQST'		be stored in the AL SECTION Cathe held.
WHERE RECNAME = 'PSPRCSRQST' OR RECNAME = 'PSPNLFIELD'	I	be stored in the AL_SECTION cache held.

CREATE THE APPLICATION ENGINE PROGRAM

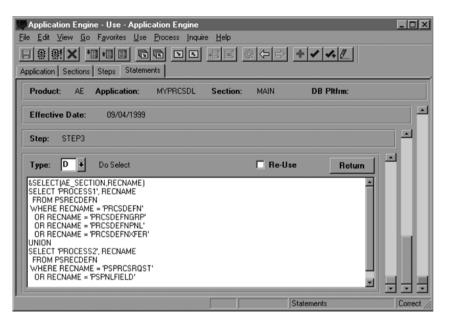


Figure 43.38 The DO Select statement to process the six process definition tables

The SQL DO Select statement above may appear strange at first. Let's talk about what we're trying to accomplish. Our goal is to delete any process definitions that match the process type and process name entered on the Run Control panel. There are six process definition tables that may contain the process definition entered. We're going to produce a result set that contains the record name (RECNAME) for each of the six tables along with the name of the dynamic section to use. Each row returned will then be processed by the appropriate section. PROCESS1 handles the tables with the PS\_ prefix, and PROCESS2 handles the tables without the PS\_ prefix. Let's test our statement using a database tool outside of PeopleSoft, SQL\*Talk in this example.

Figure 43.39 shows the Select statement results using SQL\*Talk. The first four tables use the PROCESS1 section while the last two tables use the PROCESS2 section. For each of these rows, the cache fields AE\_SECTION and RECNAME are updated. The section DYNSECTN is performed for each row, which in turn calls either the PROCESS1 or PROCESS2 sections dynamically, based on the contents of the AE\_SECTION cache field.

Our Application Engine program is finally complete! We can test our new utility to delete obsolete process definitions. It's perfectly normal to feel a bit nervous or excited before testing your work. We've put considerable effort into this application, and now we'll see if it has paid off. Let's begin our test.

W Untitled - SQLTalk for Windows		_ 🗆 ×
File Edit Session View Options Help		
SELECT 'PROCESS1', RECNAME		
FROM PSRECDEFN WHERE RECNAME = 'PRCSDEFN'		
OR RECNAME = 'PRCSDEFNGRP'		
OR RECNAME = 'PRCSDEFNPNL'		
OR RECNAME = 'PRCSDEFNXFER'		
UNION SELECT 'PROCESS2', RECNAME		
FROM PSRECDEFN		
WHERE RECNAME = 'PSPRCSRQST'		
OR RECNAME = 'PSPNLFIELD'		
ORDER BY 1,2;		
'PROCESS1' RECNAME		<b></b>
PROCESS1 PRCSDEFN PROCESS1 PRCSDEFNGRP		
PROCESSI PRCSDEFNGRP PROCESSI PRCSDEFNPNL		
PROCESS1 PRCSDEFNXFER		
PROCESS2 PSPNLFIELD		
PROCESS2 PSPRCSRQST		_
6 ROWS SELECTED		
	Idle 1 hrdmo s	

Figure 43.39 Testing our statement using SQL\*Talk

# 43.2 **TESTING THE COMPLETED APPLICATION**

Figure 43.40 shows the navigation to our new utility panel. Add a new Run Control ID. I'm going to use a Run Control ID called "DUMMY" since we're going to delete the DUMMY process definition we created earlier in this chapter.

Utilities			
Ele Edk View Go Fgvorkes Use	<u>S</u> ystem Audit <u>C</u> hange Base Language	Delete PRCSDEFN + AE Message Log +	Add Update/Display

Figure 43.40 Accessing the Delete Process Definition panel

TESTING THE COMPLETED APPLICATION

We've entered the process type and process name used for our DUMMY definition (figure 43.41). Remember, we created a DUMMY definition to test our process. We don't want to delete a "real" process definition.

Utilities - Process - Delete Process Definition	_ 🗆 ×
<u>Eile Edit View Go Favorites Use Process Help</u>	
Delete PRCSDEFN AE Message Log	
Operator Id: PS Run Control ID: DUMMY	-
Request Number: 1	
Processing Parameters	
Process Type: Crw Online +	
Process Name: DUMMY +	
Delete PRCSDEFN	L Add

Figure 43.41 Assigning parameter values on the Run Control panel

Save the record and click on the Traffic Light to initiate a Process Scheduler request. Figure 43.42 shows the Process Request panel. Our Application Engine program (MYPRCSDL) appears in the panel. Our process definition for MYPRCSDL appears to be functioning correctly, so click OK to initiate the process within Process Scheduler.

This is a good sign! You may notice the MS-DOS box appears when running on the client (figure 43.43). The lines displayed may move very quickly on the screen. Figure 43.43 shows some of the lines displayed in the MS-DOS box. Near the top of the screen, I can see our processing parameters in the Run Control record. Near the bottom, we see the first record (PRCSDEFN) had one row with our DUMMY process definition. Also notice the dynamic section PROCESS1 was performed as planned.

Process Scheduler Request		×
Operator ID: PS Run C	ontrol ID: DUMMY	
- Run Location	Output Destination	
Client     C Server	• File • Printer • Window	
Server:	File/Printer: %temp%\	OK
Run Date/Time:	Run Recurrence	Cancel
Date: 09/04/99		
Time: 05:52:00 PM	Name:	
Reset to current Date/Time	New Update Delete	
	Name Process Type Descr	
Delete Process Definition	MYPRCSDL Application Engine	
1	I	_

Figure 43.42 Submitting the Process Scheduler request

Application Engine: MYPRCSDL	-O×
17.52.57 .(AE.MYPRCSDL.MAIN.STEP2) (UPDATE)	
> 1999-09-04-17.53.00.030000 INFO(20001,2> PI(68> Program(MYPRCSDL)	
Crw Online DUMMY	
17.53.01 . <ae.myprcsdl.main.step3> <do ae.myprcsdl.dynsectn=""></do></ae.myprcsdl.main.step3>	
17.53.03(AE.MYPRCSDL.DYNSECTN.STEP1> (DO AE.MYPRCSDL.&SECTION)	
17.53.03 (AE.MYPRCSDL.PROCESS1.STEP1) (SELECT)	
17.53.04 (AE.MYPRCSDL.PROCESS1.STEP2) (DO AE.MYPRCSDL.DELETE1)	
17.53.07(AE.MYPRCSDL.DELETE1.STEP1) (UPDATE)	
17.53.08(AE.MYPRCSDL.PROCESS1.STEP3) (DO AE.MYPRCSDL.MESSAGE)	
17.53.09(AE.MYPRCSDL.MESSAGE.STEP1) (UPDATE)	
> 1999-09-04-17.53.09.860000 INFO(20001,3) PI(68> Program(MYPRCSDL)	
PRCSDEFN contains 1 records	

Figure 43.43 The MS-DOS box appears for our process

We still need to verify that the process functioned correctly, especially when a Delete statement is being executed. When the process has completed, we'll click on the A/E Message Log Folder tab to view the message log.

Figure 43.44 shows the Message Log panel. Click on the flashlight to view the messages from the latest run. We see the Run Control parameters (Crw Online DUMMY) and each table with the number of rows processed. Let's use our database query tool again to see if the rows have been deleted.

	w <u>G</u> o F <u>a</u> vo	lete Process Definition rites Use Process Help I I I I I I I II III rsage Log	
Operator:	PS	Run Control ID: DUMMY	
Application AE MYPRO		O Use Latest Process Instance     O Use Previous	
Time	Severity	Message	18
17.52.55	10	1 Requests found for PS.DUMMY	
17.52.56	10	Executing request 1 of 1	
17.52.56	10	Beginning Application AE.MYPRCSDL Delete Process Definitions	
17.53.00	10	Crw Online DUMMY	
17.53.09	10	PRCSDEFN contains 1 records	
17.53.18	10	PRCSDEFNGRP contains 2 records	
17.53.24	10	PRCSDEFNPNL contains 1 records	
Process Instance	e: 68	AE Message Log	Add //

Figure 43.44 Reviewing the message log

#### 43.2.1 Verifying our results

We verified that the DUMMY process definition rows were removed from the process definition tables using SQL\*Talk (figure 43.45). Each Select returned zero rows. We can also examine the trace file for a more detailed look at the results.

You'll find the trace file in the %TEMP%/ps/<databasename> directory. The filename <process\_instance>.aet will be used. In our particular case that translates to

```
C:\windows\temp\ps\hrdmo\68.aet
```

疑 Untitled - SQLT alk for Windows				_	
<u>File Edit Session View Options H</u> el	p				
			1		
select * from ps_prcsdefn where prcs select * from ps_prcsdefngrp where p					
select * from ps_prcsdefnpnl where p select * from ps_prcsdefnxfer where					
select * from psprcsrqst where prcsn select * from psprlield where prcsna	ame = 'DUM	IMY';			
select - from psphilleid where prosha	me = DUM	MT;			
PRCSTYPE	PRCSNAME	VERSION PARMLIS	т		
A DOTTO OPT DOPED					- 11
0 ROWS SELECTED					
PRCSTYPE	PRCSNAME	PRCSGRP			
				-	
0 ROWS SELECTED					
PRCSTYPE	PRCSNAME	PNLGRPNAME			
0 ROWS SELECTED					- 11
NONS SELECTED					
PRCSTYPE	PRCSNAME	XFERCODE MENUNAME		BARNAME	
X I					
Ready				Idle 1 hrdmo sysadm	6

Figure 43.45 Verifying our test results using SQL\*Talk

#### 43.2.2 Examining the trace file

Let's take a look at the trace file contents:

#### Listing 43.1

#### The trace file

```
17.52.56 1999-09-04 PeopleTools 7.5 Application Engine
17.52.56 Tracing request PS.DUMMY
17.52.56 Starting application AE.MYPRCSDL Delete Process Definitions
1
INSERT INTO PS_USER_AET ( PROCESS_INSTANCE, COUNTER, RECNAME, FIELDNAME,
AE_DECIDE, PRCSTYPE, PRCSNAME, AE_SECTION )
VALUES (
                 68,0,'','','','','','','')
/
COMMIT
17.52.56 .(AE.MYPRCSDL.MAIN.STEP1) (SELECT)
1
SELECT B.PRCSTYPE, B.PRCSNAME
 FROM PS_AE_RUN_CONTROL A,
                                         PS_MY_RUN_CNTL_AE B
WHERE A.OPRID = B.OPRID
  AND A.RUN_CNTL_ID = B.RUN_CNTL_ID
  AND A.PROCESS_INSTANCE = 68
```

TESTING THE COMPLETED APPLICATION

```
/
17.52.57 .(AE.MYPRCSDL.MAIN.STEP2) (UPDATE)
17.53.01 .(AE.MYPRCSDL.MAIN.STEP3) (DO AE.MYPRCSDL.DYNSECTN)
SELECT 'PROCESS1', RECNAME
  FROM PSRECDEFN
WHERE RECNAME = 'PRCSDEFN'
   OR RECNAME = 'PRCSDEFNGRP'
    OR RECNAME = 'PRCSDEFNPNL'
    OR RECNAME = 'PRCSDEFNXFER'
  UNION
 SELECT 'PROCESS2', RECNAME
 FROM PSRECDEFN
 WHERE RECNAME = 'PSPRCSROST'
   OR RECNAME = 'PSPNLFIELD'
  ORDER BY 1, 2
/
17.53.03 .. (AE.MYPRCSDL.DYNSECTN.STEP1) (DO AE.MYPRCSDL.&SECTION)
17.53.03 ... (AE.MYPRCSDL.PROCESS1.STEP1) (SELECT)
/
SELECT COUNT(*)
  FROM PS_PRCSDEFN
WHERE PRCSTYPE = 'Crw Online'
  AND PRCSNAME = 'DUMMY'
/
17.53.04 ... (AE.MYPRCSDL.PROCESS1.STEP2) (DO AE.MYPRCSDL.DELETE1)
/
SELECT 'X'
 FROM PSLOCK
WHERE 1 > 0
/
17.53.07 .... (AE.MYPRCSDL.DELETE1.STEP1) (UPDATE)
/
DELETE
 FROM PS_PRCSDEFN
WHERE PRCSTYPE = 'Crw Online'
  AND PRCSNAME = 'DUMMY'
/
17.53.08 ... (AE.MYPRCSDL.PROCESS1.STEP3) (DO AE.MYPRCSDL.MESSAGE)
17.53.09 .... (AE.MYPRCSDL.MESSAGE.STEP1) (UPDATE)
17.53.11 .(AE.MYPRCSDL.MAIN.STEP3) (DO FETCH)
17.53.11 .. (AE.MYPRCSDL.DYNSECTN.STEP1) (DO AE.MYPRCSDL.&SECTION)
17.53.12 ... (AE.MYPRCSDL.PROCESS1.STEP1) (SELECT)
/
SELECT COUNT(*)
 FROM PS_PRCSDEFNGRP
WHERE PRCSTYPE = 'Crw Online'
  AND PRCSNAME = 'DUMMY'
/
17.53.14 ... (AE.MYPRCSDL.PROCESS1.STEP2) (DO AE.MYPRCSDL.DELETE1)
/
SELECT 'X'
 FROM PSLOCK
```

```
WHERE 2 > 0
1
17.53.15 .... (AE.MYPRCSDL.DELETE1.STEP1) (UPDATE)
/
DELETE
 FROM PS_PRCSDEFNGRP
WHERE PRCSTYPE = 'Crw Online'
  AND PRCSNAME = 'DUMMY'
/
17.53.17 ... (AE.MYPRCSDL.PROCESS1.STEP3) (DO AE.MYPRCSDL.MESSAGE)
17.53.17 .... (AE.MYPRCSDL.MESSAGE.STEP1) (UPDATE)
17.53.19 .(AE.MYPRCSDL.MAIN.STEP3) (DO FETCH)
17.53.19 .. (AE.MYPRCSDL.DYNSECTN.STEP1) (DO AE.MYPRCSDL.&SECTION)
17.53.19 ... (AE.MYPRCSDL.PROCESS1.STEP1) (SELECT)
/
SELECT COUNT(*)
 FROM PS PRCSDEFNPNL
WHERE PRCSTYPE = 'Crw Online'
  AND PRCSNAME = 'DUMMY'
/
17.53.21 ... (AE.MYPRCSDL.PROCESS1.STEP2) (DO AE.MYPRCSDL.DELETE1)
/
SELECT 'X'
 FROM PSLOCK
WHERE 1 > 0
/
17.53.22 .... (AE.MYPRCSDL.DELETE1.STEP1) (UPDATE)
/
DELETE
 FROM PS_PRCSDEFNPNL
WHERE PRCSTYPE = 'Crw Online'
  AND PRCSNAME = 'DUMMY'
17.53.23 ... (AE.MYPRCSDL.PROCESS1.STEP3) (DO AE.MYPRCSDL.MESSAGE)
17.53.23 .... (AE.MYPRCSDL.MESSAGE.STEP1) (UPDATE)
17.53.25 . (AE.MYPRCSDL.MAIN.STEP3) (DO FETCH)
17.53.25 .. (AE.MYPRCSDL.DYNSECTN.STEP1) (DO AE.MYPRCSDL.&SECTION)
17.53.26 ... (AE.MYPRCSDL.PROCESS1.STEP1) (SELECT)
/
SELECT COUNT(*)
 FROM PS_PRCSDEFNXFER
WHERE PRCSTYPE = 'Crw Online'
  AND PRCSNAME = 'DUMMY'
17.53.27 ... (AE.MYPRCSDL.PROCESS1.STEP2) (DO AE.MYPRCSDL.DELETE1)
/
SELECT 'X'
 FROM PSLOCK
WHERE 1 > 0
/
17.53.28 .... (AE.MYPRCSDL.DELETE1.STEP1) (UPDATE)
/
```

TESTING THE COMPLETED APPLICATION

```
DELETE
 FROM PS PRCSDEFNXFER
WHERE PRCSTYPE = 'Crw Online'
  AND PRCSNAME = 'DUMMY'
/
17.53.29 ... (AE. MYPRCSDL, PROCESS1.STEP3) (DO AE. MYPRCSDL, MESSAGE)
17.53.30 .... (AE.MYPRCSDL.MESSAGE.STEP1) (UPDATE)
17.53.31 .(AE.MYPRCSDL.MAIN.STEP3) (DO FETCH)
17.53.31 .. (AE.MYPRCSDL.DYNSECTN.STEP1) (DO AE.MYPRCSDL.&SECTION)
17.53.32 ... (AE.MYPRCSDL.PROCESS2.STEP1) (SELECT)
SELECT COUNT(*)
 FROM PSPNLFIELD
WHERE PRCSTYPE = 'Crw Online'
  AND PRCSNAME = 'DUMMY'
/
17.53.42 ... (AE.MYPRCSDL.PROCESS2.STEP2) (DO AE.MYPRCSDL.DELETE2)
/
SELECT 'X'
 FROM PSLOCK
WHERE 0 > 0
1
17.53.43 ... (AE.MYPRCSDL.PROCESS2.STEP3) (DO AE.MYPRCSDL.MESSAGE)
17.53.44 .... (AE.MYPRCSDL.MESSAGE.STEP1) (UPDATE)
17.53.45 .(AE.MYPRCSDL.MAIN.STEP3) (DO FETCH)
17.53.45 .. (AE.MYPRCSDL.DYNSECTN.STEP1) (DO AE.MYPRCSDL.&SECTION)
17.53.46 ... (AE.MYPRCSDL.PROCESS2.STEP1) (SELECT)
SELECT COUNT(*)
 FROM PSPRCSRQST
WHERE PRCSTYPE = 'Crw Online'
  AND PRCSNAME = 'DUMMY'
/
17.53.47 ... (AE.MYPRCSDL.PROCESS2.STEP2) (DO AE.MYPRCSDL.DELETE2)
/
SELECT 'X'
 FROM PSLOCK
WHERE 0 > 0
/
17.53.48 ... (AE.MYPRCSDL.PROCESS2.STEP3) (DO AE.MYPRCSDL.MESSAGE)
17.53.48 .... (AE.MYPRCSDL.MESSAGE.STEP1) (UPDATE)
17.53.49 .(AE.MYPRCSDL.MAIN.STEP3) (DO FETCH)
1
DELETE
 FROM PS_USER_AET
WHERE PROCESS_INSTANCE = 000000068
17.53.50 Application AE.MYPRCSDL ended normally
/
COMMIT
17.53.50 Application Engine ended normally
```

In this entire trace file for the run, you can see each section and step as it was executed as well as the compiled SQL statements. Take special note of the dynamic sections PROCESS1 and PROCESS2. Also, notice the resolved bind variables used in the run.

Our Application Engine development is complete. There is always a great feeling of accomplishment that accompanies the successful completion of an application. I would suggest taking a nice long break before moving on to chapter 44 (Additional topics). You deserve some relaxation after a job well done!

### KEY POINTS

- 1 You should always determine the program structure before creating your program. Building the lower (subordinate) sections first and working backward will alleviate any step dependencies you may encounter.
- **2** Validate your program results thoroughly using your database query tool and the trace file output. Pay close attention to the resolved bind variables.
- **3** You may not have realized it, but you've taken some huge strides in learning one of PeopleSofts' up-and-coming tools, Expand your knowledge and experience by creating custom Application Engine programs to perform a variety of different tasks.



Additional topics

- 44.1 Using trace files 943
- 44.2 Restarting an A/E process 946
- 44.3 Analyzing A/E programs 947
- 44.4 Application Engine analyzer 948

So far, we've covered the basics of Application Engine, and we can surely begin developing batch processes. But what if your program does not yield the desired results? Or, worse yet, what if the process aborts unexpectedly. In this chapter, we'll discuss the use of trace files and learn how to restart an aborted process. In addition, we'll provide tips on tackling large, often cumbersome Application Engine programs. An SQR utility to help analyze A/E programs will be covered as well.

# 44.1 USING TRACE FILES

In chapter 35, we discussed the options available on the Application Definition panel. One of the options controls the creation of a trace file. The option indicates levels of detail to be included in the trace file. NO trace file is generated when the Trace option is set to Off. The trace filename is set to cprocess\_instance>.AET and is placed in the current working directory. The trace file is simply an ASCII text file that displays each step as it is executed. Generated SQL may be written to the trace file.

Figure 44.1 shows the Application Definition panel. The Trace file option is turned OFF. The other trace options available are SQL and Steps Only. SQL displays all the steps executed along with the associated SQL statements. The Steps Only option only displays the steps executed. (No SQL statements are written to the trace file.)

Application Engine - Use - Application Engine File Edit View Go Favorites Use Process	<u>I</u> nquire <u>H</u> elp	© </th <th>_ 🗆 X</th>	_ 🗆 X
Application Sections Steps Statements			
Product: AE Application: SAMPL	E		
Description:		_	
Cache Record:	¥		
Version:			
Message Set + Number:			
Trace: Off	As Of Date		
Debug	Use current		
Issue Message for each Commit	Override	01/01/1900	
Force Commit after each Step	Last Updated		
Force Abend after each Commit	by:		
Disable Restart	at:		
Rebuild SQL Statements			
		Application	Add

Figure 44.1 Trace options on Application Definition panel

#### 44.1.1 Sample trace file

This sample trace file was generated during an execution of our exercise 5 application (called AE.USER005). The trace option was set to SQL, which displays all the steps and SQL statements processed. You can see some of the SQL statements are controlled by the PTPEMAIN process:

08.28.54 1999-01-13 PeopleTools 7.5 Application Engine 08.28.54 Tracing request PS.#USER005 08.28.54 Starting application AE.USER005 User Application 005

USING TRACE FILES

```
/
INSERT INTO PS_USER_AET ( PROCESS_INSTANCE,COUNTER,RECNAME,FIELDNAME,
AE_DECIDE )
VALUES ( 10,0,'','','')
/
```

The Insert statement immediately preceding is an example of a PTPEMAINcontrolled statement. It initializes the cache record we specified on the Application Definition panel. Fields are initialized depending on their datatype. PROCESS\_INSTANCE is always set to the process instance of our program. The field, COUNTER, is initialized to zero since it is numeric. The remaining fields in the cache record are initialized to blank since they are character datatypes.

```
UPDATE PS_USER_AET
   SET FIELDNAME = 'PAY_END_DT'
WHERE PROCESS_INSTANCE = 0000000010
/
COMMIT
/
```

The statement above is also generated by PTPEMAIN. Remember, we initialized the cache field FIELDNAME to a value of 'PAY\_END\_DT' on the Process Request panel. The value is then loaded into our cache record. This is how process request parameters are passed to the program.

The step in the code below selects RECNAME values and for each row returned (or fetched) executes a section called COUNT. The resolved bind variable for FIELD-NAME is displayed in the trace file, and the value is set to 'PAY\_END\_DT'.

The RECNAME passed to COUNT.STEP1 (in this instance) is BEN\_PLAN\_DATA:

```
08.28.58 ..(AE.USER005.COUNT.STEP1) (SELECT)
/
SELECT COUNT(*)
FROM PS_BEN_PLAN_DATA
/
08.29.00 ..(AE.USER005.COUNT.STEP2) (DO AE.USER005.MSG)
/
SELECT 'X'
```

```
FROM PSLOCK
WHERE 0 > 0
/
```

Notice all &BIND() variables have been resolved in the SQL. STEP1 selects the row count for the BEN\_PLAN\_DATA table. COUNT.STEP2, as you may recall, performs a DO When statement. If the number of rows is greater than zero, the section MSG is performed. The WHERE clause against the PSLOCK table may look odd at first. In our statement definition, the WHERE clause was defined as WHERE &BIND(COUNTER) > 0. Since the cache field COUNTER contains a value of zero, the statement was compiled as WHERE 0 > 0. This returns NO rows, and the MSG section is not executed. Control once again is passed to MAIN.STEP1, and another row is fetched:

```
08.29.02 .(AE.USER005.MAIN.STEP1) (DO FETCH)
08.29.03 ..(AE.USER005.COUNT.STEP1) (SELECT)
/
SELECT COUNT(*)
FROM PS_BOND_LOG
/
08.29.05 ..(AE.USER005.COUNT.STEP2) (DO AE.USER005.MSG)
/
SELECT 'X'
FROM PSLOCK
WHERE 471 > 0
/
08.29.07 ...(AE.USER005.MSG.STEP1) (UPDATE)
```

This time a RECNAME value of BOND\_LOG is passed to the COUNT section. There were 471 rows in the table. The DO When criteria once resolved reads as WHERE 471 > 0. This returns a row from PSLOCK, which designates a TRUE condition. The section MSG is then executed:

```
08.29.09 .(AE.USER005.MAIN.STEP1) (DO FETCH)
08.29.10 ..(AE.USER005.COUNT.STEP1) (SELECT)
/
SELECT COUNT(*)
FROM PS_DED_CALC
/
08.29.13 ..(AE.USER005.COUNT.STEP2) (DO AE.USER005.MSG)
/
SELECT 'X'
FROM PSLOCK
WHERE 90 > 0
/
08.29.15 ...(AE.USER005.MSG.STEP1) (UPDATE)
/
```

This process continues until no rows remain. We'll skip the rest and proceed to the end of the trace file:

```
/
(
08.38.24 .(AE.USER005.MAIN.STEP1) (DO FETCH)
/
DELETE
FROM PS_USER_AET
WHERE PROCESS_INSTANCE = 0000000010
/
08.38.24 Application AE.USER005 ended normally
/
COMMIT
/
08.38.25 Application Engine ended normally
```

The last DO FETCH returned no rows, so the COUNT section was no longer executed. Our defined program has completed, and PTPEMAIN does some final cleanup by deleting the cache record row we've been using.

## 44.2 RESTARTING AN A/E PROCESS

During the execution of your A/E program, an unexpected error may cause it to abort. There are numerous reasons why this may occur: for example, an SQL error, system resource problem, or syntax errors in your A/E statement. A critical process could be near completion when the error occurs, and starting the process over from the beginning may not be a feasible solution. Application Engine maintains an entry in the AE\_RUN\_CONTROL table, which holds restart information in the event of an abend. This restart information is refreshed at every commit point. When restarting, the process takes over from the last commit point and continues. You cannot submit an aborted process from the beginning using the same OPRID and RUN\_CNTL\_ID. The AE\_RUN\_CONTROL holds it in a suspended status so it may be restarted properly.

Only applications run on the server through Process Scheduler can be restarted using Process Monitor by highlighting the failed process and clicking on Action  $\rightarrow$  Restart. All other applications must execute PTPEMAIN.exe manually on the client or server when restarting. PTPEMAIN.exe should reside in the CBLBIN subdirectory attached to the PSVER directory (PSVER standing for the PeopleSoft version assigned as the high-level directory name). Simply type PTPEMAIN on the command line. You will then be prompted for the database type, database name, username (OPRID), password, Run Control ID, and process instance. The process will then resume where it left off.

There are times when you may wish to start the process from the beginning. If so, you'll have to delete the AE\_RUN\_CONTROL row for the OPRID and RUN\_CNTL\_ID you're using. You may also have to delete the cache record for the process instance. Only then can you restart the process from the beginning. You may also disable the restart capability on the Application Definition panel. Please make certain that no data corruption can occur as a result of not restarting properly.

# 44.3 ANALYZING A/E PROGRAMS

Analyzing large Application Engine programs can be tedious. One section can call a multitude of other sections. Since all components are stored within the database, you'll have to toggle back and forth between sections using the online panels. You can, however, take several steps that will make your analysis a little easier:

- Read any documentation on the process beforehand.
- Print a listing of the cache record used by the A/E program. This identifies all fields used to pass values from one step to another.
- Start with the section MAIN. Identify all steps within the MAIN section and treat each step as a separate process. Breaking the process down into smaller logical sections will help put things in perspective.
- Keep track of everything that's happening, not just the relationship between sections but also the tables being affected. Make a list of permanent tables and temporary tables. The temp tables can pass large amounts of data to subsequent sections or steps.
- After running the process, look at the trace file produced. One method of trace execution I've seen used allows you to sort the lines in the trace file using an ASCII editor (or import the lines into Excel or Access and then sort them). To do so, you remove any lines that don't have the time-stamp on it. You are then left with the time and name of all the steps performed in execution order. Be wary of processes that run beyond midnight. You'll have to manipulate the file in your editor to correct the sequencing. The resulting data will look like this:

```
08.28.54 1999-01-13 PeopleTools 7.5 Application Engine
08.28.54 Starting application AE.USER005 User Application 005
08.28.54 Tracing request PS.#USER005
08.28.55 .(AE.USER005.MAIN.STEP1) (DO AE.USER005.COUNT)
08.28.58 ..(AE.USER005.COUNT.STEP1) (SELECT)
08.29.00 ..(AE.USER005.COUNT.STEP2) (DO AE.USER005.MSG)
08.29.02 .(AE.USER005.COUNT.STEP1) (DO FETCH)
08.29.03 ..(AE.USER005.COUNT.STEP1) (SELECT)
08.29.05 ..(AE.USER005.COUNT.STEP2) (DO AE.USER005.MSG)
08.29.07 ...(AE.USER005.COUNT.STEP2) (DO AE.USER005.MSG)
08.29.07 ...(AE.USER005.MSG.STEP1) (UPDATE)
08.29.09 .(AE.USER005.MAIN.STEP1) (DO FETCH)
08.29.10 ..(AE.USER005.COUNT.STEP1) (SELECT)
08.29.11 ...(AE.USER005.COUNT.STEP1) (SELECT)
08.29.13 ...(AE.USER005.COUNT.STEP2) (DO AE.USER005.MSG)
08.29.15 ...(AE.USER005.MSG.STEP1) (UPDATE)
```

#### We'll skip the middle part of the trace and go to the end.

08.38.19 .(AE.USER005.MAIN.STEP1) (DO FETCH) 08.38.20 ..(AE.USER005.COUNT.STEP1) (SELECT) 08.38.22 ..(AE.USER005.COUNT.STEP2) (DO AE.USER005.MSG) 08.38.24 .(AE.USER005.MAIN.STEP1) (DO FETCH) 08.38.24 Application AE.USER005 ended normally 08.38.25 Application Engine ended normally This method gives you a good idea of the execution flow of the program but it's not one hundred percent accurate. The first three lines were all processed at 08.28.54, but they are not in execution order. The third line, "Tracing request PS.#USER005," should come before the "Starting application...." line. Any lines with the same time-stamp will then sort alphabetically. When analyzing a specific portion of the program, this may not be a factor.

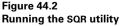
- A simple SQR program can be written to extract the time-stamped lines from the trace file. This would be more effective than manipulating the file in Excel or Access. Basic SQR skills would be required.
- When using a trace file, remember a trace file only displays the steps based on certain conditions met at the time. Running the process two different times could follow two different execution paths. The trace files are simply "after the fact." PeopleSoft does not provide a mechanism to produce an indented "tree-formatted" flowchart of an entire A/E process. The next portion of this chapter deals with a custom SQR utility I developed to analyze an Application Engine program. The cache record is listed; each step is listed with all of its defined attributes; and, finally, an indented process flowchart is produced which illustrates the full execution flow of the process. I have named this utility (appropriately) "Application Engine Analyzer."

# 44.4 APPLICATION ENGINE ANALYZER

I developed this SQR utility to extract all the required information to analyze an A/E program. Once the program is identified through user-entered prompts, several functions are performed. The cache record is listed; the steps are printed with their attributes; and an indented process flowchart is produced.

When running the SQR utility, make sure the communication box is visible by using the -CB option (figure 44.2). This displays additional information before each prompt.

SQR V4.2.3		
Report name         c:\sqr\td_ae75.sqr           Database         epdmo	<u>O</u> K Cancel	
Username sysadm Password Report arguments:	<u>Files</u>	
<pre>-ic:\sqr\;c:\fdm75su\user\sqr\;c:\ FDM75SU\sqr\ -mc:\FDM75SU\sqr\allmaxes.max -oC:\TEMP\sqr.log -2IFc:\FDM75SU\sqr\Pssqr.ini -lockr1 -CB -fC:\TEMP\ -s</pre>	About	Figure 44.2



The basic prompts are product, application ID and database platform. Figure 44.3 shows an example of the prompts issued when running the Application Analyzer utility. The product entered is AR, which is the product ID for Accounts Receivable. The application ID is PREDICT, the Payment Predictor process found in Accounts Receivable. This example was run under SQLBASE, so I entered the SQL-BASE code of "1" for the database platform prompt. When a step has a SQLBASE-specific version, the utility substitutes that in place of the generic one. This is how Application Engine processes steps as well. An additional prompt creates an A/E statement file, which produces a temporary file that can be imported into Word, formatted, and printed. It includes extracted A/E statements also in execution order. This is extremely valuable information put in an organized fashion. Only use this feature when absolutely necessary—the output can be very large, slowing the process down.



Figure 44.3 SQR utility prompts for user information

The user is also prompted when a dynamic DO section (&SECTION) is encountered. By entering the section #DETAIL, the dynamic DO is resolved and included in the process. #DETAIL is a payment predictor matching algorithm that may be called dynamically when running the payment predictor process. You may substitute any valid algorithm for the &SECTION substitution variable.

Let's now examine some sample output which has been condensed for the book:

Report ID: Phase:	_		APPLICAT	TION ENG	INE ANALY	ZER
Program:	AR/PREDICT	Г	Cache I	Record:	PP_AET	
Fieldname		Кеу	Туре	Len	Dec	LongName
=======================================	===========	:	=========		=======	
PROCESS INS	TANCE	Y	Nbr	10		Process Instance
OPRID		N	Char	8		Operator Id
RUN_CNTL_ID		N	Char	30		Run Control ID
SETID		Ν	Char	5		SetID
PP_METHOD		N	Char	15		Payment Predictor Method
EFFDT		N	Date	10		Effective Date
PP_SEQ_NUM		Ν	Nbr	3		Sequence
PP_USAGE		Ν	Char	1		Usage
PP_SORT_SEQ	_NUM	N	Nbr	3		Sorting Sequence number

The AR.PREDICT Application Engine program (known as Payment Predictor) uses a cache record called PP\_AET. The first portion of the utility prints each field in PP\_AET along with the field attributes.

The second portion lists each step in the A/E program with the MAIN section listed first. Additional attributes, such as DO section, Step Information, DO types, DB Platform, and so forth, are listed:

Report I Phas Progra	e: 2		AP	PLICATION ENGINE	ANALY	YZEF	{		
Section	Step	Do	Act	ivity	===== Tj	==== /ре	Update	Select	DO_When
=======	=========	===========	====	=======================================	=====	====			======
MAIN	INIT	INIT	(DO	INIT)	D		N	N	N
MAIN	PREP	PREP	(DO	PREP)	D		Ν	Ν	Y
MAIN	SBLD	SBLD	(DO	SBLD)	D		Ν	Ν	Y
MAIN	UPDM	UPDM	(DO	UPDM)	D		N	Ν	Y
MAIN	DOC_SEQ	DOCSEQ	(DO	DOCSEQ)	D		N	Ν	Ν
MAIN	REALGAIN	REALGAIN	(DO	REALGAIN)	D		Ν	Ν	Ν
MAIN	PGEN	PGEN	(DO	PGEN)	D		N	Ν	Y
MAIN	PUPD	PUPD	(DO	PUPD)	D		N	Ν	Y
MAIN	TERMINAT	TERMINAT	(DO	TERMINAT)	D		Ν	Ν	Ν

Let's look at steps that execute a COBOL process and a dynamic DO section. Notice the &SECTION has been resolved with the user-entered section #DETAIL.

DOC-CBL	FTPDOCAE		(COBOL:	FTPDOCAE)	С	Ν	Ν	Ν
STEP	ALGR	#DETAIL	(DO &SE	CTION: #DETAIL)	D	N	N	Y

The process flow portion of the SQR utility begins with the section MAIN and flowcharts all steps in execution order. If a section has already been analyzed, the literal <Repeated Section> appears after it. There is no need to drill-down a second time:

```
Report ID: TD_AE75
                    APPLICATION ENGINE ANALYZER
  Phase: 3
 Program: AR/PREDICT
_____
Process Flowchart
_____
.... (MAIN.INIT) (DO INIT)
.....(INIT.STARTMSG)
                  (UPDATE)
.....(INIT.ROUNDSET)
                  (SELECT)
                  (DO ROUNDIN3)
.....(INIT.ROUNDIN3)
(SELECT)
.....(INIT.REQUESTS)
                  (UPDATE)
.....(INIT.CNT)
              (SELECT)
.....(INIT.MSG)
              (UPDATE)
.....(INIT.NONE)
              (DO MSG NONE)
(UPDATE)
.... (MAIN.PREP)
           (DO PREP)
..... (PREP.CLEARTMP)
                  (DO CLEARTMP)
..... (CLEARTMP.PAYMENT)
                       (UPDATE)
.....(CLEARTMP.CUST)
                    (UPDATE)
..... (CLEARTMP.ITEM)
                     (UPDATE)
(UPDATE)
..... (CLEARTMP.MATCH)
                      (UPDATE)
.....(PREP.MESSAGE) (UPDATE)
.....(PREP.PAYMENTS)
                 (UPDATE)
..... (PREP.DOC SEO)
                (UPDATE)
.....(PREP.SET_REF)
                 (UPDATE)
.....(PREP.ID_ITEM)
                 (DO ID_ITEM)
.....(ID_ITEM.CUSTMP1)
                       (UPDATE)
(DO DUPECUST)
(UPDATE)
(UPDATE)
.....(DUPECUST.CLEANUP)
                           (UPDATE)
.....(ID_ITEM.CUSTMP2)
                       (UPDATE)
.....(ID_ITEM.DUPE2) (DO DUPECUST)
                                 <Repeated Section>
```

Here is the portion of the process flowchart where the dynamic DO is encountered. The #DETAIL section entered by the user has been substituted and included in the listing:

.....(STEP.ALGR) (DO &SECTION: #DETAIL) .....(#DETAIL.ALGO\_1) (SELECT) .....(#DETAIL.ADJUST) (DO ADJUST) .....(ADJUST.INIT) (UPDATE) ......(ADJUST.ADJ\_OVER) (UPDATE) <Repeated Section> .....(#DETAIL.MATCHTMP) (UPDATE) .....(#DETAIL.ALGR B1) (DO DUPES) .....(DUPES.DUPES) (DO ALGR\_DUP) (SELECT) .....(#DETAIL.ALGR C1) (DO PYSTATUS)

Let's look at how the COBOL section appears in the process flowchart. The developer can easily identify any COBOL or Mass Change sections:

```
....(MAIN.DOC_SEQ) (DO DOCSEQ)
.....(DOCSEQ.DOC_SEQ) (DO DOC_SEQ)
.....(DOC_SEQ.CHK_SEQ) (DO DOC-CBL)
.....(DOC_SEQ.CHK_SEQ) (DO DOC-CBL)
.....(DOC_SEQ.SETID) (SELECT)
.....(DOC_SEQ.GET_TYPE) (SELECT)
.....(DOC_SEQ.UPD_BU) (UPDATE)
.....(DOC_SEQ.GET_SEQ) (DO DOC-CBL) <Repeated Section>
....(MAIN.REALGAIN) (DO REALGAIN)
```

#### 44.4.1 Application Engine Analyzer source code – TD\_AE75.SQR

The Application Engine Analyzer program processes sections and steps in the same manner as the PTPEMAIN process. The complete source code may be downloaded from the website http://www.sqrtools.com (under Utilities).

This process has been tested under Oracle, SQLBase, and DB2, but it may work with other databases as well. Additional updates may be posted to SQRTOOLS.COM, which may include compatibility with non-compliant databases.

Versions prior to Application Engine 7.5 are supported as well. Simply deactivate the substitution variable AE\_75. This will bypass all references to the columns AE\_DO\_PRODUCT and AE\_DO\_APPLID. A major (and quite useful) enhancement in version 7.5 was the ability to call sections outside of the current application. The two aforementioned columns allow a called section to be qualified with the product and application ID, if necessary.

### **KEY POINTS**

- 1 Use the Trace option to generate trace files for the Application Engine program. The trace file will show you the steps performed along with the SQL statements and resolved bind variables.
- 2 You can restart an Application Engine program so it picks up at the last commit point before it failed. This helps maintain system integrity when a process aborts.



Application Engine-PeopleSoft 8

45.1 Application Engine "wish list" 95545.2 PeopleSoft release 8 955

As we have proceeded through each Application Engine chapter, we've covered more concepts of Application Engine development. As an SQL processing tool, A/E can be used to create efficient batch processes. A/E's many useful features include decision capability and loop control, dynamic section calling, and messaging functionality. Other nice features in the current release are the trace file generation and the Commit/Restart logic (when a process terminates abnormally). All said, A/E is an extremely useful and well-conceived tool.

# 45.1 APPLICATION ENGINE "WISH LIST"

Although Application Engine is undeniably a tribute to creativity and resourcefulness, one can't help but think of enhancements that might still be made to the existing product. For instance, Application Engine exclusively acts on data that resides in the database itself. Imagine if Application Engine had the capability to read or write external files. This would make Application Engine an ideal choice for interface and conversion applications.

The Application Engine Definition panels are adequate for developing your programs, but a more intuitive graphical interface would be more suitable. The ability to view your program as a tree structure with each section and step in execution order would be a tremendous help.

There are times when updating a simple cache field value may seem cumbersome. Using the &SELECT function against the PSLOCK table (or any other single row table) is an ingenious solution, but is a bit convoluted. It also requires an additional call to the database where an alternative method may not need to do so. Application Engine could also use a mechanism to handle complex IF-THEN-ELSE expressions.

The ability to add Application Engine components to a project would be a welcome enhancement. Customizations could then be managed the same as other PeopleTool objects. Also, having Change Control in effect to lock your Application Engine programs would prevent other users from concurrently updating your program.

The types of enhancements I've mentioned here would elevate Application Engine to a much higher level, making it difficult to ignore the batch-processing capability that Application Engine provides. Let's now take a look at some of the great features implemented in release 8, some of which are nothing short of spectacular.

# 45.2 **PEOPLESOFT RELEASE 8**

Release 8 of PeopleSoft contains all of our "wish list" enhancements plus many additional features that can make Application Engine the tool of choice for many business processes. The single most important feature is Application Engines' complete integration with Application Designer. All Application Engine components are now objects. This means they can be placed into projects just as a record or panel definition would. You can also utilize the Change Control functionality to lock and unlock the Application Engine objects you're working on. Application Engine is now written entirely in C++. COBOL is no longer used to execute A/E programs.

When creating or modifying Application Engine objects, you will encounter a new graphical interface. It is much more intuitive than prior versions and behaves in a fashion similar to PeopleTool object interfaces. The Application Engine program is displayed in Definition or Program Flow view. The Program Flow view allows you to view your Application Engine program as a tree structure with each section and step displayed as a tree node. You can click on the "+" or "-" to expand or collapse a node. Any object type actions within a step are also displayed. Object types include SQL

Selects (DO types), SQL objects, other A/E sections, Message Log, and PeopleCode objects. Yes, that's correct—PeopleCode! Application Engine can now invoke PeopleCode and share many common business functions with online PeopleCode. A/E can be used to update fields in a state record (formerly referred to as a cache record). Any complex IF-THEN-ELSE expressions can be written in PeopleCode as well.

A new set of PeopleCode functions and classes have been added to support Application Engine. Some allow the reading and writing of external files. A new PeopleCode File class has been created that allows a variety of file handling operations to take place. You can even define a file layout with the new file layout definition in Application Designer and utilize it in your program.

Application Engine functions and macros have been replaced with Meta-SQL and a new set of macros. The Meta-SQL set has been expanded for greater functionality. For example, system (Meta) variables, which serve as text substitution variables, have been introduced. An example of a Meta-Variable would be <code>%ProcessInstance</code>, which contains the process instance of the run. <code>%RunControl</code> contains the Run Control ID used for the run. Prior versions of Application Engine required a database call against the A/E Run Control table to retrieve these values.

Temporary tables used in Application Engine programs have also increased functionality. Application Designer allows you to specify if a table is temporary. If so, you may designate the number of temporary table instances. For example, if you have a record called MY\_TEMP, defined as a temporary table with three instances, the following physical SQL tables are created: PS\_MY\_TEMP, PS\_MY\_TEMP01, PS\_MY\_TEMP02, and PS\_MY\_TEMP03. During the execution of the Application Engine program, a specific instance of the temporary table can be utilized. This can greatly improve efficiency when running parallel processes.

Another interesting feature added is the Access checkbox on the section properties. If the section is designated as Public, all external Application Engine programs may call the section. If it is not Public, then the section is not available to any other programs. This is an excellent security measure that will prevent sensitive and potentially destructive SQL statements from being executed inadvertently.

The Application Engine debugger is also introduced in PeopleSoft release 8. While using the debugger, you can set break points, step through the code, view and edit state record fields, and even switch to the PeopleCode debugger when executing PeopleCode actions. This is a great feature that will make testing and debugging your Application Engine programs much easier than in the past.

Let's take a quick tour of some of the Application Engine features in PeopleSoft release 8.

### 45.2.1 Application Designer—Creating

lew	OK
App Engine Program	
Approval Rule Set	Cancel
Business Component	
Business Interlink	-

Figure 45.1 Creating a new Application Engine program

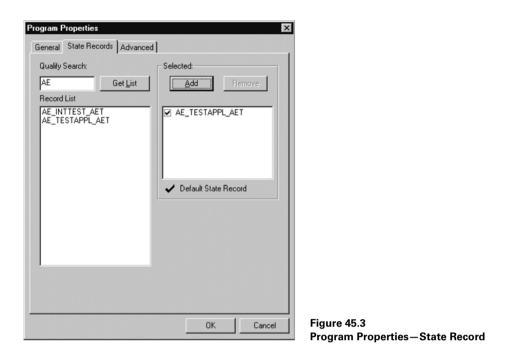
As mentioned previously, the Application Engine Designer Interface is accessed through Application Designer. You can create an Application Engine program by selecting File  $\rightarrow$ New... and then selecting the Application Engine Program object in the drop down list (figure 45.1).

Once you select the new or existing Application Engine program, you can view or modify

the program properties. Figure 45.2 shows the Program Properties panel. You can add a description and comments on the General folder tab.

Program Properties	
General State Records Advanced	
28	
Description:	
Comments:	
×	
Owner ID:	
Last Updated Date/Time: By Operator:	
OK Cancel	Figure 45.2 Program Properties – General

The State Record folder tab allows you to enter the State record(s) used by your Application Engine program. Multiple state records may be utilized by the program. You set the default state record by clicking on the Default State Record checkbox (figure 45.3). Note that the state record was formerly known as a cache record. The state record must still end with the suffix \_AET as in prior versions.



State records have much more functionality in release 8. They can now be used globally. The same state record can be used by both the calling and called program. Parameters can easily be passed from one program to another when sharing the same state record.

The Advanced folder tab (figure 45.4) lets you specify the default Message Set, Disable Restart, and designate Upgrade Only programs. In addition, your program can be defined as an Application Library. An Application Library is not an executable Application Engine program but a collection of sections that can be called by other Application Engine programs.

You make your actual program modifications using the Application Engine Definition interface. Two tabs allow you to view your program components: the definition view and the program flow view. The definition view (figure 45.5) allows you to create sections, steps, and actions, which are displayed as nodes. You can collapse and expand the nodes to drill down into each section. The sections in the definition view are not displayed in the order they are executed. You need to click on the Program Flow tab to view the execution order of the program.

Pay close attention to the Project Workspace window in figure 45.5. The Application Engine object has been inserted into the project. As you can see, Application Engine is fully integrated with PeopleTools in release 8.

Program Properties		×
General State Records	Advanced	1
Disable <u>R</u> estart		
Application Library	Message <u>S</u> et: 0	•
Vpgrade Only		
	OK	Cancel



Application Designer - PT800TEST - [VAC0]			_ D ×
		Ø <u>₹</u> + =_	
PT800TEST	Definition Program Flow	•	
Application Engine Programs	Section	Step Action	<u>^</u>
	🛨 🗀 MAIN	Convert Virtual Approver table	MAIN.GBL.(
	RELTBLS	Section1 description	RELTBLS.G
	RELTBLS	Section1 description	RELTBLS.G
	RELTBLS	Section1 description	RELTBLS.G
	RELTBLS	Section1 description	RELTBLS.G
	RELTBLS	Section1 description	RELTBLS.G
	RELTBLS	Section1 description	RELTBLS.C
Development     的     Development	(		
1 Application Engine Programs object(s) inserted. ≪ ▶  Build  Find Object References  Upgrade	Results / Validate /		×
Ready			

Figure 45.5 Application Engine Definition view

You can access the section properties by clicking on the section node and then clicking the right mouse button. Notice the Access checkbox in the section properties (figure 45.6). You can make the section Public by clicking the checkbox. Another

🔁 MAIN	MAIN desci	ription		MAIN.GBL.(base,	).1900-01-01		
	Market:	Platform:	Effective Date:	Effective Status:	Section Type:	Auto Commit:	Access:
	GBL	(base)	01/01/1900	Active	Prepare Only	After Step	Public

Figure 45.6 Viewing section properties

new feature is the Market designation. You can define your section as Global (GBL) or use a market code such as USA or JPN to make your section market specific.

The Application Engine Program Flow View (figure 45.7) displays the program as a tree structure with each node in its logical execution sequence. This feature should aid developers by providing a graphical representation of their Application Engine program.

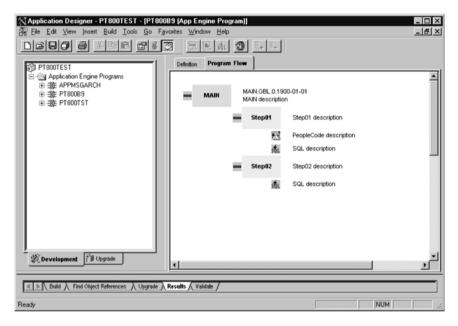


Figure 45.7 Application Engine Program Flow view

Notice the PeopleCode node under Step01 (figure 45.7). You can insert People-Code actions (in Definition view) within a step. You use the PeopleCode Editor to write the PeopleCode program.

You can invoke the PeopleCode Editor by double-clicking on the PeopleCode node (Figure 45.8). SQL actions can be inserted the same way within a step and modified using something called the SQL Editor. Each Action Type, viewed as a node, will have a particular set of action type properties.

PeopleCode	Vertify Number of Rows	
	On Return:	
	Skip Step	

Figure 45.8 Accessing a PeopleCode program in Definition view

Figure 45.9 shows two action type nodes. The Do Select action type displays the description, Reuse statement, and Do Select type properties. The Call Section action type displays the description, section name, program ID, and dynamic section properties.

Do Select	Do Select descri	Do Select description			
	ReUse Statement	Do Select Type:			
	No	Select/Fetch			
Call Section	Call Section des	cription			
	Section Name:	Program ID:			
	STATS	AETESTPROG	Dynamic		

Figure 45.9 Action types and action type properties

### 45.2.2 Action types

Table 45.1 shows the possible action types along with the object type, the available properties, and the corresponding editor used to create the code behind the action. COBOL and Mass Change programs can no longer be called through the action properties. You can still call a COBOL program, but it must be invoked using PeopleCode and the RemoteCall() function. Mass Change programs are no longer supported—alternatives, such as the Application Engine Mass Change program, can be used instead. The message action is used in place of the &MSG function. The Message properties contain the same parameters as the &MSG function.

Table 45.1 Action types and associated properties

Action Type	Object Type	Properties	Editor
DO When	SQL Select	ReUse Statement	SQL Editor
DO While	SQL Select	ReUse Statement	SQL Editor
DO Until	SQL Select	ReUse Statement	SQL Editor
DO Select	SQL Select	ReUse Statement DO Select Type	SQL Editor
PeopleCode	PeopleCode	On Return	PeopleCode Editor
SQL	SQL Statement	ReUse Statement No Rows	SQL Editor

Action Type	Object Type	Properties	Editor
Call Section	A/E Section	Section Name Program ID Dynamic Section	N/A
Message	Message Log	Message Set Message Set Num- ber Message Parame- ters	N/A

 Table 45.1
 Action types and associated properties (continued)

#### 45.2.3 Meta-SQL

Application Engine now supports Meta-SQL such as %DateIn and %DateOut. Application Engine Meta-SQL constructs have been added.

#### Table 45.2

Function	Description
%Bind	Retrieves a value from the State record.
%ExecuteEdits	Supports data dictionary edits in batch mode. This includes any field defined with edit types of Required, Yes/No, DateRange, Prompt Table, OT Trans- late Table. Meta-Variables %Edit_Required, %Edit_YesNo, %Edit_DateRange, %Edit_PromptTable, and %Edit_TranslateTable are used to specify the particular Edit(s) required. These Meta-Variables can be added together to produce combination edits on a field.
%Select	Selects fields and updates State Record values. If the SQL select returns no rows, the state record fields are untouched.
%SelectInit	Selects fields and updates state record values. If no rows are returned, the state record fields are initialized.
%SQL	Allows an SQL object to be utilized in Application Engine SQL statements or PeopleCode regardless of differences in bind variable syntax between the two.
%Table	Returns the SQL table name for the record name specified. This eliminates the need to prefix certain tables with PS_before accessing them. If the table is defined as a temporary table, the appropriate temporary table instance number is appended to the returned SQL table name (i.e., PS_MY_TEMPnn where nn is the instance number).
%TruncateTable	Depending on the database, either a TRUNCATE TABLE OF DELETE FROM (with- out a where clause) is generated.
%UpdateStats	Generates a platform-specific statement to update the system catalog tables for use in optimization procedures.

### 45.2.4 Application Engine macros

Some macros in release 8 look familiar. Some of the differences are in syntax only. The &&RECORD macro is no longer used.

Table 45.3 Application Engine macros

Macro	Description
%ClearCursor	recompiles re-used statements. Resets any STATIC %Bind variables
%Execute	execute database-specific commands such as PL/SQL Blocks
%Next	increments a sequence value
%Previous	decrements a sequence value
%RoundCurrency	rounds an amount to proper currency precision when using the Multi-Currency option

### 45.2.5 System Meta-Variables

Application Engine now provides useful Meta-Variables that eliminate unnecessary calls to the database to retrieve fields such as the Run Control ID and process instance. The SQL syntax is also simpler when using the Meta-Variables. %Bind is not needed to retrieve the values from the state record.

Meta-Variable	Description
%AeProgram	current Application Engine program name (in quotes)
%AeSection	current Application Engine section name (in quotes)
%AeStep	current Application Engine step name (in quotes)
%JobInstance	Process Scheduler job instance number
%ProcessInstance	Process Instance
%ReturnCode	return code of last SQL statement
%RunControl	current Run Control ID (in quotes)
%AsOfDate	As-Of-Date of the current process (in quotes)
%Comma	character substitution— comma
%LeftParen	character substitution—left parenthesis
%RightParen	character substitution—right parenthesis
%Space	character substitution—space
%SQLRows	number of rows affected by SQL statement. Select statements return a value of 0 or 1 (to represent no rows or some rows, respectively).

Table 45.4 Application Engine System Variables

### 45.2.6 Application Engine PeopleCode

The use of PeopleCode is one of the most powerful enhancements to Application Engine. You can update state records directly, perform complex IF-THEN-ELSE expressions, and process file input/output records. When you attach a PeopleCode

Action to a step, you may also specify the On-Return property, which is either Abort, Break, or Skip Step. The On-Return property is initiated when the PeopleCode program issues a Non-Zero or True return code. If no return code is assigned by the PeopleCode program, then zero is used as the default. Abort halts processing of the entire program. Break exits the entire section currently executing. Skip Step processes no additional actions attached to the current step—the next step is processed immediately.

Let's examine a few examples, drawing comparisons with PeopleSoft 7.5 when possible.

You can see that a database call is required in PeopleSoft 7.5 to update a single cache field value:

```
&SELECT (AE_SECTION)
SELECT 'PROCESS1'
FROM PSLOCK
```

In the direct updating of the state record field using PeopleCode (PeopleSoft 8.0), no additional database call is required. Also, note the absence of a return code assignment which defaults to zero:

```
AE_SECTION = "PROCESS1";
```

Let's consider IF-THEN-ELSE logic now. Imagine that we need to execute a Mass SQL Insert only if the table into which we're inserting is empty. For this example, we assume the row count of our table into which has been determined and is contained in the field COUNTER (either in the cache or state record).

In PeopleSoft 7.5, a DO When statement type is used to execute an additional section, depending on the "SQL Select" results. If the COUNTER cache field is zero (meaning the table is empty), the DO When section specified (which performs the SQL Insert) is executed. If the COUNTER is not zero, the DO When section is not performed:

```
&SELECT (AE_DECIDE)
SELECT `X'
FROM PSLOCK
WHERE &BIND(COUNTER) = 0
```

Look at the PeopleSoft 8 version using PeopleCode:

```
If USER_AET.COUNTER > 0
   Exit(1)
End-if;
```

The state record field COUNTER is interrogated directly. If the COUNTER field is greater than zero, the return code is set to 1 (or TRUE), and the On Return property of the PeopleCode action becomes effective. Let's assume the On Return property is set to Skip Step. Any additional actions for the current step are now bypassed

including the SQL action for our Insert statement. If the COUNTER is zero, the subsequent SQL action for the step is executed. (The Return Code defaults to zero in our PeopleCode action.)

PeopleSoft 8 enables file operations within Application Engine through PeopleCode. This is made possible by the new object classes now available. We'll demonstrate how PeopleCode actions can use these object classes to write records to a flat file. The output file will be created using a file layout definition. By changing the File-Layout property of our file object, we can switch file layouts whenever necessary.

We can now perform a simple demonstration for a typical outbound interface program. The sample PeopleCode program creates a flat file based on the contents of the table MY\_TABLE. Let's assume this table was created during preceding steps of the Application. The columns are selected using a temporary SQL object created dynamically at run time. A Meta-SQL function (%Selectall) is used to build the Select statement. The SQL Object uses the Fetch method to retrieve each row one at a time. The file object (our output file) uses the file layout definition MY\_LAYOUT:

```
Ln# PeopleCode
____ ____
1 Local Record &MY REC;
2 Local File &MY_FILE;
  Local SQL &MY_SQL;
3
Δ
5
  &MY_FILE = GetFile("myoutput.txt", "W");
6
7 if &MY_FILE.IsOpen Then
8
     if &MY_FILE.SetFileLayout(FILELAYOUT.MY_LAYOUT) Then
9
        &MY REC = CreateRecord(RECORD.MY TABLE);
10
       &MY SOL = CreateSOL("%Selectall(:1)", &MY REC);
       While &MY_SQL.Fetch(&MY_REC)
11
12
          &MY_FILE.WriteRecord(&MY_REC);
13
      End-While;
14 End-If;
15 End-If;
16
17
  &MY FILE.Close();
18
```

Our program is displayed above with line numbers (for reference only). Let's take a closer look at each line in the PeopleCode program.

Lines 1 through 3 create temporary object variables: &MY\_REC is a record object; &MY\_FILE is a file object; and &MY\_SQL is an SQL object. Each of these temporary variables has a set of properties unique to its own object type. We can now perform some simple manipulations to accomplish our task.

Line 5 uses the GetFile function to associate a file to our file object &MY\_FILE. The GetFile function also opens "myoutput.txt" in Write Mode.

Line 7 tests to see if the file associated with &MY\_FILE was opened successfully (evaluating the IsOpen property using dot notation).

Line 8 sets the FileLayout property of the &MY\_FILE file object to our file layout definition (MY\_LAYOUT).

Line 9 uses the CreateRecord function to pass the MY\_TABLE attributes to the &MY\_REC record object. Now MY\_TABLE and &MY\_REC have equivalent attributes.

Line 10 dynamically creates the SQL for the &MY\_SQL object using the CreateSQL function. %Selectall(:1) is a Meta-SQL construct that creates the Select statement based on the record passed as a parameter. Since we passed the &MY\_REC record object as the parameter, the record MY\_TABLE is used. (Remember, &MY\_REC now has the same attributes as MY\_TABLE.)

Lines 11 through 13 perform a Do While loop. A Fetch method is performed using the SQL object we created (&MY\_SQL). This selects each row one by one. The WriteRecord method for the &MY\_FILE object is used with the &MY\_REC object to write lines to the output file. The records are then written as directed by the file layout definition currently used.

The operations being performed may be considered complex, but the PeopleCode that is actually produced by the developer couldn't be much simpler. Also, note any changes to the original record or file layout definitions do not affect the PeopleCode program.

#### 45.2.7 Application Engine debugger

Another exciting enhancement in release 8 is the Application Engine debugger. You must enable the debugger through Configuration Manager or as a command line option. You must also enable the PeopleCode debugger if you want to debug any PeopleCode actions in the Application Engine program. Debug mode is easy to use. Here's a glimpse of the Application Engine debugger Help menu:

```
PeopleTools 8.0 - Application Engine
Copyright (c) 1988-1999 PeopleSoft, Inc.
All Rights Reserved
Application Engine Debugger - enter command or type ? for help.
AEMYPRCSDL.MAIN.STEP1> ?
Debug Commands:
       (Q)uit
                       Rollback work and end program
      E(X)it
                       Commit work and end program (valid between steps)
                       Commit work (valid between steps)
       (C)ommit
       (B)reak
                       Set or remove a break point
       (L)ook
                      Examine state record fields
       (M)odify
                      Change a state record field
       (W)atch
                       Set or remove a watch field
       (S)tep over
                       Execute current step or action and stop
                       Go inside current step or called section and stop
  Step (I)nto
```

CHAPTER 45 APPLICATION ENGINE—PEOPLESOFT 8

Step	(0)ut of	Execute rest of step or called section and stop
	(G) o	Resume execution
	(R)un to commit	Resume execution and stop after next commit

As you can see, the Application Engine debugger contains an extensive set of debug commands. The descriptions of the commands on the Help menu do not need much more elaboration, but I'd like to review a couple of commands.

The BREAK command allows you to set, and subsequently unset, breakpoints in your program. There is also an option to list the currently active breakpoints.

Let's consider how to set a breakpoint with the Set option. The user is prompted for the program, section, and step to which the breakpoint should be set:

```
AEMYPRCSDL.MAIN.STEP1> b
(S)et, (U)nset, or (L)ist? s
Program [AEMYPRCSDL]:
Section [MAIN]: DYNSECTN
Step [STEP1]: STEP1
Breakpoint set at AEMYPRCSDL.DYNSECTN.STEP1
```

Now, let's look at the use of the Unset option. A list of active breakpoints is displayed along with a corresponding sequence number. The user must enter the sequence number of the breakpoint to remove it from the active breakpoint list. The List option displays the active breakpoint list without any additional options.

```
AEMYPRCSDL.MAIN.STEP1> b
(S)et, (U)nset, or (L)ist? u
Active Breakpoints:
(1) AEMYPRCSDL.MAIN.STEP2
(2) AEMYPRCSDL.DYNSECTN.STEP1
Remove which breakpoint? 1
```

The LOOK, MODIFY, and WATCH commands allow you to view and modify state record fields and designate watch fields. Once you set a watch field, the program stops when the value of the field changes.

```
Record Name [USER_AET]:

Field Name [*]:

USER_AET:

PROCESS_INSTANCE = 50

COUNTER = 1685

RECNAME = 'JOB'

FIELDNAME = ' '

AE_DECIDE = ' '
```

In our LOOK command, the record name selected was USER\_AET (the default for this Application Engine program). All the fields in USER\_AET are listed with their current values.

Consider now the results of our MODIFY command. We selected the COUNTER state record field and changed the value from 1685 to 0. The MODIFY command is a useful tool when testing conditions in your Application Engine program:

```
AEUSER003.MAIN.STEP2> m
Record Name [USER_AET]:
Field Name [none]: COUNTER
Current value: USER_AET.COUNTER = 1685
Enter new value (do not use quotes around text strings):
0
```

The field RECNAME is selected as a watch field using the Set option of the WATCH command. The program will stop each time the value of this field changes. You can Unset and List watch fields in a similar manner as breakpoints.

```
Set or remove a watch field
AEUSER004.MAIN.STEP1> w
(S)et, (U)nset, or (L)ist? s
Record Name [USER_AET]:
Field Name [none]: RECNAME
```

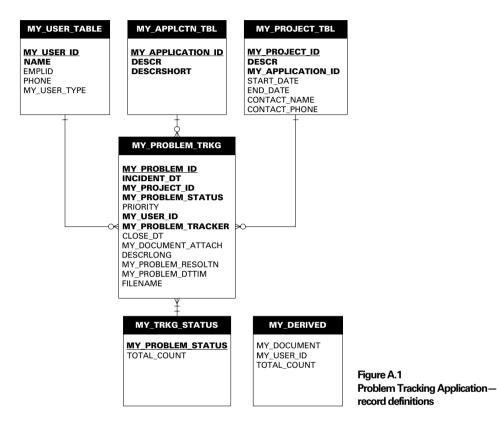
If enabled, the PeopleCode debugger is invoked when a PeopleCode action is encountered. Many new features exist in the PeopleCode debugger such as Hover Inspect, where a pop-up displays the value of simple variables and fields simply by hovering over it with the mouse. The variable display window allows you to drill down on the properties of each object by expanding/collapsing the corresponding node. As you can see, the PeopleTools development team has been busy. The features I've mentioned in this chapter are just a small sampling of the next generation of PeopleTools!

Some readers may have come directly to this chapter to read about some of the great new Application Engine features in the PeopleSoft 8 release. If you have no familiarity with the PeopleSoft 7.5 version of Application Engine, I would suggest going through the tutorial in the preceeding chapters. The examples there will give you the opportunity to develop a good understanding of Application Engine concepts without being bombarded with terminology such as object classes, meta-this, meta-that, and such. The PeopleSoft 8 version of Application Engine builds and improves upon the concepts previously discussed. In this ever-changing world of technology, it's a good idea to take advantage of every learning opportunity you can.

#### APPENDIX A

Problem Tracking application

All objects used to build our Problem Tracking application are listed in this appendix. The readers can develop these objects as they read this book. The readers should not be limited to the objects in the appendix. They can further enhance the application and develop other objects by using the techniques described in this book. Let us look at an ERD diagram of all the record definitions used to develop the Problem Tracking application. All columns which are in bold and underlined are part of the primary key. All columns in bold alone are alternate keys.



#### **Problem Tracking-Record Definitions**

Let us list all the record definitions showing the different views. New fields start with a prefix of MY\_. These fields have to be created in the system before the record definitions are built.

Field Name	Туре	Len	Format	н	Short Name	Long Name
MY USER ID NAME EMPLID PHONE MY_USER_TYPE	Char Char Char Char Char	6 50 11 24 1	Upper Name Upper Custm Upper		Name ID Phone	<u>User ID</u> Name EmpIID Telephone User Type

Figure A.2 MY\_USER\_TABLE Table – Field Display

ali MY_USER_TABLE (Record)											
Field Name	Туре	Key	Dir	CurC	Srch	List	Sys	Audt	н	Default	
MY USER ID	Char	Key	Asc	[	Yes	Yes	No	[	İ		
NAME EMPLID PHONE	Char Char Char		Asc Asc		No No No	Yes Yes No	No No No				
MY_USER_TYPE	Char				No	No	No				

Figure A.3 MY\_USER\_TABLE Table—Use Display

灣 MY_USER_TABLE	(Record)					_ 🗆 ×
Field Name	Type F	Req Ec	dit	Prompt Table	Set Control Field	Rs Dt
MY USER ID	Char Y	'es 🛛			Í	No
NAME EMPLID PHONE MY_USER_TYPE	Char N Char N	res No Pre No Xia	ompt at	PERSONAL_DATA		No No No
•				'	'	. ·

Figure A.4 MY\_USER\_TABLE Table – Edits Display

MY\_USER\_TABLE stores all the users reporting problems in our application.

∰ MY_APPLCTN_TBL (F						
Field Name	Type	Len	Format	H	Short Name	Long Name
MY APPLICATION ID	Char	3	Upper	l	Application	Application Identification
DESCR	Char	30	Mixed			Description
DESCRSHORT	Char	10	Mixed		Short Desc	Short Description

Figure A.5 MY\_APPLCTN\_TBL Table—Field Display

編 MY_APPLCTN_TBL (Record)											
Field Name	Туре	Key	Dir	CurC	Srch	List	Sys	Audt	н	Default	
MY APPLICATION ID	Char	Кеу	Asc	[	Yes	Yes	No	[			
DESCR DESCRSHORT	Char Char	Alt	Asc		No No	Yes No	No No				

Figure A.6 MY\_APPLCTN\_TBL Table—Use Display

Field Name	Type	Req	Edit	Prompt Table	Set Control Field	Rs D
MY APPLICATION ID	Char	Yes		1	[	No
DESCR DESCRSHORT	Char Char					No No

Figure A.7 MY\_APPLCTN\_TBL Table—Edits Display

MY\_APPLCTN\_TBL stores all applications that are tracked in our system.

慧 MY_PROJECT_TBL (Re	ecord)					
Field Name MY_PROJECT_ID DFSCB	Type Char Char	Len 6 30	Format Upper Mixed	H	Short Name Project ID Descr	Long Name Project Identification Description
MY_APPLICATION_ID START_DATE	Char Date	3 10	Upper		Application Start Date	Application Identification Start Date
END_DATE CONTACT_NAME CONTACT_PHONE	Date Char Char	10 50 12	Mixed Phone		End Date Name Phone	End Date Contact Name Contact Phone

Figure A.8 MY\_PROJECT\_TBL Table – Field Display

護 MY_PROJECT_TBL (F	Record)									
Field Name	Туре	Key	Dir	CurC	Srch	List	Sys	Audt	Н	Default
MY PROJECT ID DESCR MY_APPLICATION_ID START_DATE END_DATE CONTACT_NAME CONTACT_NAME CONTACT_PHONE	Char Char Date Date Char Char	Key Alt	Asc Asc Asc	1		Yes Yes No No No	No No No No No No	]		

Figure A.9 MY\_PROJECT\_TBL Table—Use Display

謄 MY_PROJECT_TBL (F	lecord)					_ 🗆 ×
Field Name	Type F	Reg	Edit	Prompt Table	Set Control Field	Rs Dt
MY PROJECT ID	Char  Y			[	]	No
DESCR MY_APPLICATION_ID START_DATE END_DATE CONTACT_NAME CONTACT_PHONE	Char N Date N Date N Char N	lo lo	Prompt	MY_APPLCTN_TBL		No No No No No
•				·		•

Figure A.10 MY\_PROJECT\_TBL Table—Edits Display

MY\_PROJECT\_TBL stores all projects that are tracked in our application.

標編 MY_PROBLEM_TRKG(	Recor	d)				
Field Name	Туре	Len	Format	Н	Short Name	Long Name
MY PROBLEM ID	Char	6	Num		Problem ID	Problem Identification
INCIDENT_DT	Date	10		[	Incdnt Dt	Incident Date
	Char	6	Upper		Project ID	Project Identification
MY_PROBLEM_STATUS	Char	1	Upper		Problem Sta	Problem Status
PRIORITY	Nbr	3			Priority	Priority
MY_USER_ID	Char	6	Upper		User ID	User ID
MY_PROBLEM_TRACKEF	Char	6	Upper		Problem Tra	Problem Tracker
CLOSE_DT	Date	10			Close Date	Date Closed
MY_DOCUMENT_ATTACI	Char	1	Upper		Document?	Document Attached?
DESCRLONG	Long	0			Descr	Description
MY_PROBLEM_RESOLT	Long	0			Prob.Resolu	Problem Resolution
MY_PROBLEM_DTTIM	DtTm	26	Scnds		Date/Time	Date/Time Reported
FILENAME	Char	80	Mixed		File Name	File Name

Figure A.11 MY\_PROBLEM\_TRKG Table – Field Display

Field Name	Туре	Key	Dir	CurC	Srch	List	Sys	Audt	Η	Default
MY PROBLEM ID	Char	Кеу	Asc		Yes	Yes	No			
INCIDENT_DT	Date	Alt	Asc	1	No	Yes	No			
	Char	Alt	Asc		No	Yes	No			
MY_PROBLEM_STATUS	Char	Alt	Asc		No	Yes	No			'1'
PRIORITY	Nbr				No	No	No			
MY_USER_ID	Char	Alt	Asc		No	Yes	No			
MY_PROBLEM_TRACKEF	Char	Alt	Asc		No	Yes	No			
CLOSE_DT	Date				No	No	No			
MY_DOCUMENT_ATTACI	Char				No	No	No			
DESCRLONG	Long		1		No	No	No			
MY_PROBLEM_RESOLTI	Long				No	No	No			
MY <sup>PROBLEM</sup> DTTIM	DtTm				No	No	No			
FILENAME	Char				No	No	No			

Figure A.12 MY\_PROBLEM\_TRKG Table—Use Display

Field Name	Туре	Req	Edit	Prompt Table	Set Control Field	Rs DI
MY PROBLEM ID	Char	Yes		[	1	No
INCIDENT_DT	Date	Yes				Yes
	Char	No	Prompt	MY_PROJECT_TBL		No
MY_PROBLEM_STATUS	Char	No	Xlat			No
PRIORITY	Nbr	No				No
MY_USER_ID	Char	No	Prompt	MY_USER_TABLE		No
MY_PROBLEM_TRACKEF	Char	No	Prompt	MY_USER_TABLE		No
CLOSE_DT	Date	No	-			No
MY_DOCUMENT_ATTACI	Char	No	Y/N			No
DESCRLONG	Long	No				No
MY_PROBLEM_RESOLTI	Long	No				No
MY PROBLEM DTTIM	DtTm	No				No
FILENAME	Char	No				No

Figure A.13 MY\_PROBLEM\_TRKG Table—Edits Display

MY\_PROBLEM\_TRKG stores all incidents tracked in our application.

🏥 MY_TRKG_STATUS (R	ecord)					
Field Name	Туре	Len	Format	н	Short Name	Long Name
MY PROBLEM STATUS	Char	1	Upper		Problem Sta	Problem Status
TOTAL_COUNT	Nbr	7			Total Cnt	Total Count

Figure A.14 MY\_TRKG\_STATUS View—Field Display

齇 MY_TRKG_STATUS (R	ecord)									
Field Name	Туре	Кеу	Dir	CurC	Srch	List	Sys	Audt	Η	Default
MY PROBLEM STATUS	Char	Key	Asc		No	No	No	1		
TOTAL_COUNT	Nbr				No	No	No			

Figure A.15 MY\_TRKG\_STATUS View—Use Display

灣 MY_TRKG_STATUS (R	ecord)					- 🗆 ×
Field Name	Туре	Req	Edit	Prompt Table	Set Control Field	Rs Dt
MY PROBLEM STATUS	Char	No	Xlat			No
TOTAL_COUNT	Nbr	No				No

Figure A.16 MY\_TRKG\_STATUS View – Edits Display

MY\_TRKG\_STATUS is an SQL view that represents data from MY\_PROBLEM\_ TRKG table.

<b>Record Properties</b>		×
General Use Type Record Lype C SQL Table	SQL <u>V</u> iew Select Statement:	
SQL View     SQL View     Dynamic View     Derived/Work     SubRecord     Query View	my_problem_status, count(*) from ps_my_problem_trkg group by my_problem_status	
Non-Standard SQL Table Name:		T
	ОК	Cancel

Figure A.17 MY\_TRKG\_STATUS View—SQL Select Statement

	1-					
Field Name	Туре	Len	Format	<u>  H</u>	Short Name	
MY DOCUMENT	Char	1	Upper		Document E	Document Button
MY_USER_ID	Char	6	Upper		User ID	User ID
TOTAL_COUNT	Nbr	7			Total Cnt	Total Count
_	1 1		1			
	1 1					
	1 1					
	1 1					
	1 1		1			
	1 1					
	1 1		1			
	1 1		1			
	1 1		1			

Figure A.18 MY\_DERIVED Work Record—Field Display

뻅 MY_DERIVED (Red	cord)									
Field Name	Туре	Key	Dir	CurC	Srch	List	Sys	Audt	H	Default
MY DOCUMENT	Char		[	1	No	No	No	1		
MY_USER_ID	Char		1	1	No	No	No			
TOTAL_COUNT	Nbr				No	No	No			
							1			
							1			

Figure A.19 MY\_DERIVED Work Record—Use Display

ଞ୍ଜାMY_DERIVED (Red	cord)					
Field Name	Туре	Req	Edit	Prompt Table	Set Control Field	Rs DI
MY DOCUMENT	Char	No	Y/N	1	<u> </u>	No
MY_USER_ID TOTAL_COUNT	Char Nbr	No No				No No

Figure A.20 MY\_DERIVED Work Record—Edits Display

MY\_DERIVED is a derived record that holds three fields which are used as work fields in our application.

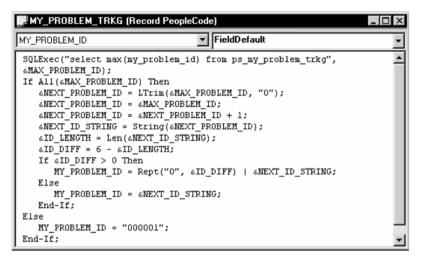


Figure A.21 MY\_PROBLEM\_TRKG.MY\_PROBLEM\_ID.FieldDefault

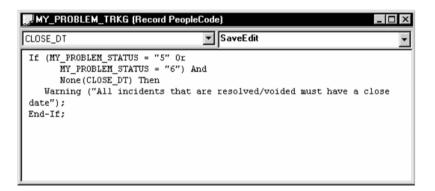


Figure A.22 MY\_PROBLEM\_TRKG.CLOSE\_DT.SaveEdit

The first PeopleCode program automatically increments the MY\_PROBLEM\_ID field to the next one. The second PeopleCode program performs an edit to ensure that the CLOSE\_DT field is entered when incidents are resolved or voided.

MY_PROBLEM_TRK	G (Record PeopleCode)	×
MY_PROBLEM_ID	SearchSave	•
Warning ("Pr	n (MY_PROBLEM_ID) Then oblem ID is automatically assigned by the system - want to override.");	

Figure A.23 MY\_PROBLEM\_TRKG.MY\_PROBLEM\_ID.SearchSave

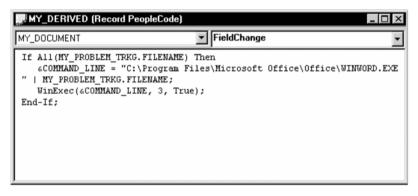


Figure A.24 MY\_DERIVED.MY\_DOCUMENT.FieldChange

The first PeopleCode program prevents the user from assigning a value to the MY\_PROBLEM\_ID field when new incidents are added using our application. The second PeopleCode program opens a Microsoft Word document explaining an incident in our application.

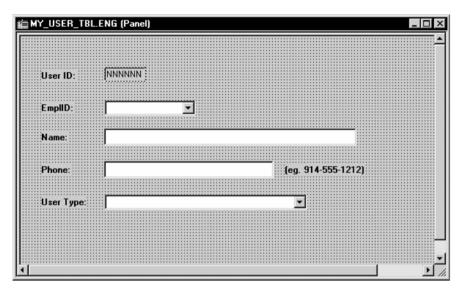


Figure A.25 MY\_USER\_TBL panel

Edit Box Edit Box Edit Box Text Drop Down List Drop Down List	MY_USER_ID NAME PHONE MY_USER_TYPE EMPLID	MY_USER_TABLE MY_USER_TABLE MY_USER_TABLE MY_USER_TABLE MY_USER_TABLE MY_USER_TABLE
	Edit Box Edit Box Text Drop Down List	Edit Box NAME Edit Box PHDNE Text Drop Down List MY_USER_TYPE

Figure A.26 MY\_USER\_TBL panel layout

MY\_USER\_TBL panel is used to enter user information on our application.

User ID		User Type	EmplID	Telephone
000110			Linpito	Totophono
_				
_				
_				
				1
				1
	1			1
				<u> </u>

Figure A.27 MY\_USER\_GRID panel

um L\	I Label	Type	Field	Record
	* * * Top of List * * *			
	User Table Grid * * * End of List * * *	Grid		MY_USER_TABLE
	ОКОС	ancel <u>S</u> e	lect Move Unse	slect Default

Figure A.28 MY\_USER\_GRID panel layout

MY\_USER\_GRID panel is used to enter user information in a grid format.

MY_APPLCTN_TBL.ENG (Panel)		
Application ID:	HIY APPLOTE TRUENG (Papel)	
Description:		
Description:	······	
Description:	Application IU:	
		~~~~~~
	Description	
	Description.	
		111111111111111111111111111111111111111
Short Description:		
Short Description:		
	Short Description	
,		

Figure A.29 MY\_APPLCTN\_TBL panel

um Lvl	Label	Туре	Field	Record
1 0 2 0 3 0	*** Top of List *** Application ID Description Short Description *** End of List ***	Edit Box Edit Box Edit Box	MY_APPLICATION_ID DESCR DESCRSHORT	MY_APPLCTN_TBL MY_APPLCTN_TBL MY_APPLCTN_TBL
	ОК Са	ancel <u>S</u> elect	Move Unselect	Default

Figure A.30 MY\_APPLCTN\_TBL panel layout

MY\_APPLCTN\_TBL panel is used to set up applications that are tracked through our Problem Tracking application.

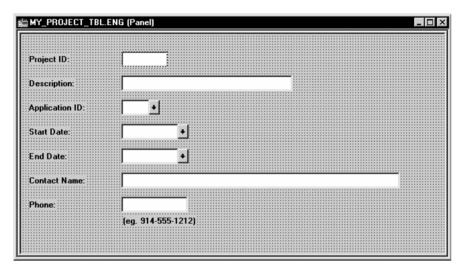


Figure A.31 MY\_PROJECT\_TBL panel

1         0         Project ID         Edit Box         MY_PROJECT_ID         MY_PROJECT_TBL           2         0         Description         Edit Box         DESCR         MY_PROJECT_TBL           3         0         Application ID         Edit Box         MY_APPLICATION_ID         MY_PROJECT_TBL           4         0         Start Date         Edit Box         START_DATE         MY_PROJECT_TBL           5         0         End Date         Edit Box         EDID_DATE         MY_PROJECT_TBL           6         0         Contact Name         Edit Box         CONTACT_NAME         MY_PROJECT_TBL	lum Lv	Label	Туре	Field	Record
8 0 (eg. 914-555-1212) *** End of List ***	2 0 3 0 4 0 5 0 6 0 7 0	Description Application ID Start Date End Date Contact Name Phone (eg. 914-555-1212)	Edit Box Edit Box Edit Box Edit Box Edit Box Edit Box Edit Box	DESCR MY_APPLICATION_ID START_DATE END_DATE	MY_PROJECT_TBL MY_PROJECT_TBL MY_PROJECT_TBL MY_PROJECT_TBL

Figure A.32 MY\_PROJECT\_TBL panel layout

MY\_PROJECT\_TBL panel is used to enter project information in our application.

MY_PROBLEM_T	222222	
Incident Date:	L Close	Date:
Project ID:	<ul> <li>•</li> <li>•</li> </ul>	AAAAAAAAAAAAAAAAAAAA
Application ID:	NNN AAAAAAAAAAAAAA	Date/Time Reported:
Status:	•	AAAAAAAAAAAAAAAAAAAAA 22/22/2222 10:22:22P2
Priority:	User ID:	
Document?	Open Tracker:	• AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
File Name:		
Problem:		
Resolution:		

Figure A.33 MY\_PROBLEM\_TRKG panel

lum Lvl		Туре	Field	Record
1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 0 9 0 10 0 11 0 11 0 13 0 14 0 15 0 16 0	*** Top of List *** Problem ID Problem Tracking Incident Date Close Date Project ID Dummy Name Application ID Description Status Dummy Name Priority User ID Name Tracker Dummy Name Open File Name	Edit Box Frame Edit Box Edit Box	MY_PROBLEM_ID INCIDENT_DT CLOSE_DT MY_PROJECT_ID DESCR MY_APPLICATION_ID DESCR MY_PROBLEM_STATUS XLATLONGNAME PRIORITY MY_USER_ID NAME MY_PROBLEM_TRACKER NAME MY_DOCUMENT FILENAME MOVE Unselect	MY_PROBLEM_TRKG MY_PROBLEM_TRKG MY_PROBLEM_TRKG MY_PROBLEM_TRKG MY_PROJECT_TBL MY_PROJECT_TBL MY_PROJECT_TBL MY_PROBLEM_TRKG XLATTABLE MY_PROBLEM_TRKG MY_PROBLEM_TRKG MY_PROBLEM_TRKG MY_PROBLEM_TRKG MY_PROBLEM_TRKG

Figure A.34 MY\_PROBLEM\_TRKG panel layout

MY\_PROBLEM\_TRKG panel is used to enter incidents and resolutions through our Problem Tracking application.

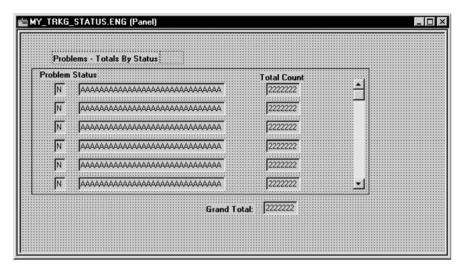


Figure A.35 MY\_TRKG\_STATUS panel

*** Top of List ***	lum Lvl	Label	Type	Field	Record
	1 0 2 0 3 0 4 1	Problems - Totals By Stat Frame Grand Total Status Problem Status Status Total Count	Frame Edit Box Scroll Bar Edit Box Edit Box	MY_PROBLEM_STATUS XLATLONGNAME	- MY_TRKG_STATUS XLATTABLE

Figure A.36 MY\_TRKG\_STATUS panel layout

MY\_TRKG\_STATUS panel is used to view totals of all incidents/problems tracked using our application.

Ē	MY_USERS.GBL	(Panel Group)			
Γ	Panel Name	ltern Name	Hidden	ltern Label	Folder Tab Label
1	MY_USER_TBL	MY_USER_TBL		Users	
4	1				

Figure A.37 MY\_USERS panel group

Panel Group Properties	×
General Use	
Access Search record: MY-USER TABLE Add search record: Detail panel: MY_USER_TBL	Actions Actions Actions Actions Actions Actions Display Update/Display Diplate/Display Diplate/Display Diplate/Display Diplate/Display Diplate/Display Diplate/Display
3-Tier Execution Location Panel Group <u>B</u> uild Client Application server Default (application server)	Panel Group Sa <u>v</u> e C Client Application server Default (application server)
	OK Cancel

Figure A.38 MY\_USERS panel group properties

É	📺 MY_USER_GRID.GBL (Panel Group)						
	Panel Name	Itern Name	Hidden	Itern Label	Folder Tab Label		
1	MY_USER_GRID	MY_USER_GRID		User Table Grid			

Figure A.39 MY\_USER\_GRID panel group

Panel Group Properties General Use	×
Access Search record: [INSTALLATION Add search record: Detail ganel: [MY_USER_GRID]	Actions Actions Actions Actions Update/Display Update/Display All Correction Data Entry
3-Tier Execution Location Panel Group <u>B</u> uild C Client C Application server C Default (application server)	Panel Group Sa <u>ve</u> C Client C Application server C Default (application server)
	OK Cancel

Figure A.40 MY\_USER\_GRID panel group properties

É	MY_APPLICATIO	NS.GBL (Panel Group	)		_ 🗆 ×
	Panel Name	ltern Name	Hidden	ltern Label	Folder Tab Label
1	MY_APPLCTN_TBL	MY_APPLCTN_TBL		Applications	

Figure A.41 MY\_APPLICATIONS panel group

Panel Group Properties	×
General Use	
Access Search record: MY_APPLCTN_TBL Agd search record: Detail ganet: MY_APPLCTN_TBL	Actions Actions Add Update/Display Update/Display All Correction Data Entry
3-Tier Execution Location Panel Group <u>B</u> uild Client C Application server Default (application server)	Panel Group Sa <u>v</u> e C Client C Application server Default (application server)
	OK Cancel

Figure A.42 MY\_APPLICATIONS panel group properties

Ľ	🗃 MY_PROJECTS.GBL (Panel Group)					
Γ	Panel Name	ltern Narne	Hidden	ltern Label	Folder Tab Label	
1	MY_PROJECT_TBL	MY_PROJECT_TBL		Projects		

Figure A.43 MY\_PROJECTS panel group

Panel Group Properties General Use	×
Access Search record: MY PROJECT TBL Add search record: Detail ganel: MY_PROJECT_TBL	Actions ✓ Add ✓ Update/Display Update/Display All ✓ Correction ✓ Data Entry
3-Tier Execution Location Panel Group <u>B</u> uild Client Application server Default (application server)	Panel Group Sa <u>v</u> e C Client Application server C Default (application server)
	OK Cancel

Figure A.44 MY\_PROJECTS panel group properties

É	📺 MY_PROBLEM_TRKG.GBL (Panel Group)						
	Panel Name	ltern Name	Hidden	ltern Label	Folder Tab Label		
1	MY_PROBLEM_TRK	MY_PROBLEM_TRKG		Problem Tracking			

Figure A.45 M\_PROBLEM\_TRKG panel group

Panel Group Properties				X
General Use				
Access Search record: MY_PROBLEM_TRKS Add search record: Detail ganet: MY_PROBLEM_TRKG	- - -	Actions Add Update/ Update/ Correction Data Enl	Display All m	
3-Tier Execution Location Panel Group Build C Client C Application server C Default (application server)	C Clie	àroup Sa <u>v</u> e — nt olication server iault (applicatio		
		OK	Cancel	

Figure A.46 M\_PROBLEM\_TRKG panel group properties

É	🗃 MY_TRKG_STATUS.GBL (Panel Group)					
	Panel Name	ltern Narne	Hidden	ltern Label	Folder Tab Label	
1	MY_TRKG_STATUS	MY_TRKG_STATUS		Problems - Totals b		

Figure A.47 MY\_TRKG\_STATUS panel group

Panel Group Properties	×
General Use	
Access Search record: INSTALLATION Add search record: Detail ganet: MY_TRKG_STATUS	Actions Actions Add Update/Display Update/Display All Correction Data Entry
3-Tier Execution Location Panel Group <u>B</u> uild C Client C Application server C Default (application server)	Panel Group Sa <u>v</u> e C Client C Application server C Default (application server)
<u> </u>	OK Cancel

Figure A.48 MY\_TRKG\_STATUS panel group properties

🖆 PROBLEM_TRACKING (Menu)
File Edit View Go Favorites       Setup Tracking       Language Help         Users       User Table Grid         Applications       Projects         Image: Setup Tracking Image Help

Figure A.49 Problem Tracking menu-setup bar items

ig PROBLE	EM_TR	ACK	ING (Men	u)			<
Eile Edit	⊻iew	<u>G</u> o	Favorites	Setup	Tracking : Irack Problem Problems - To :	Help	

Figure A.50 Problem Tracking menu-tracking bar items

Menu Properties	×
General Use	
Menu Label: Problem Tracking Menu Group Problem Tracking Define &Business Rules 9993 &Develop Workforce 1 &Administer Workforce 2 PeopleSoft 9993 &EDI Manager 9993 &EDI Manager 9993 Monitor &Workplace 4 Player Info 9993	
Seguence #: 9999 Sec Alphabetical order	nu Group Order quence #: 9999 Alphabetical order Add geparator after
	OK Cancel

Figure A.50 Problem Tracking menu properties

#### APPENDIX B

**Operator** Class/Locations

The Operator Class and Employee Locations application links PeopleSoft operator classes to office locations. Employees currently working out of these office locations can be linked to the Operator Class/Location for security and reporting purposes.

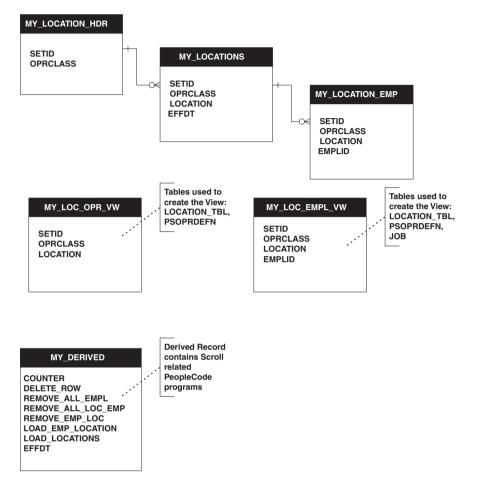
The application is comprised of three main records, two views, and a Derived/ Work record which stores the PeopleCode statements. The application is used primarily to demonstrate the use of scroll-related functions.

The records are: MY\_LOCATION\_HDR MY\_LOCATIONS MY\_LOCATION\_EMP

The two Views are: MY\_LOC\_OPR\_VW MY\_LOC\_EMPL\_VW

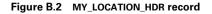
The Derived/Work record is: MY\_DERIVED

Figure B.1 graphically illustrated the records, the views, and the Derived/Work record for this Applet.





MY_LOCATION_I	IDR (Record	)												-	□×
Field Name	Туре	Key	Dir	CurC	Srch	List	Sys	Audt	н	Defa	ult				
SETID	Char			İ		Yes	No	İ				TBL	HR.	SETID	,
OPRCLASS	Char		Asc	<u> </u>	Yes	Yes	No	<u> </u>							



Field Name	Туре	Key	Dir	CurC	Srch	List	Sys	Audt	H	Default	
SETID	Char	Key	Asc	1	Yes	Yes	No	ſ	· · · · ·	OPR DEF TBL HR.9	ETID
OPRCLASS	Char		Asc	1	Yes	Yes	No	1		l	
LOCATION	Char	Key	Asc		Yes	No	No				
EFFDT	Date	-	1		No	No	No			%Date	

Figure B.3 MY\_LOCATIONS record

簡 MY_LOCATION_	EMP (Record	)											_	I ×
Field Name	Туре	Key	Dir	CurC	Srch	List	Sys	Audt	H	Default				
SETID OPRCLASS LOCATION EMPLID	Char Char Char Char Char	Key Key	Asc Asc Asc Asc Asc		Yes Yes Yes No	Yes Yes No No	No No No No			OPR DEF	TBL	HR.	SETID	



၏ MY_LOC_OPR_\	∕₩ (Record)									
Field Name	Туре	Key	Dir	CurC	Srch	List	Sys	Audt	Н	Default
SETID	Char	Key	Asc	1	Yes	Yes	No	Ī	Ì	ſ
OPRCLASS LOCATION	Char Char		Asc Asc		Yes Yes	Yes No	No No			



၏ MY_LOC_EMPL_	VW (Record)									_ 🗆 ×
Field Name	Туре	Key	Dir	CurC	Srch	List	Sys	Audt	H	Default
SETID	Char	Key					No	1		
OPRCLASS	Char	Key	Asc	1	Yes	Yes	No	1		
LOCATION	Char	Key	Asc		Yes	No	No	1		
EMPLID	Char	Key	Asc		No	No	No			

Figure B.6 MY\_LOC\_EMPL\_VW view

The PeopleCode associated with this application resides primarily in the Derived/ Work record MY\_DERIVED as illustrated in figure B.7. Refer to part 3 for PeopleCode illustrations related to scroll processing.

Field Name	Туре	FDe FEd	FCh	FFo	BIn	RIs	RDe	RSe	SEd	SPr	SPo	Srl	SrS	Wrk	PPr
	Sign		ļ		Yes		[]						[		
DELETE_ROW REMOVE_ALL_EMPL REMOVE_ALL_LOC_EMP	Char Char Char Char Char Char Char Date		Yes Yes Yes Yes Yes Yes Yes		Yes				Yes Yes						Yes

Figure B.7 MY\_DERIVED Derived/Work record

Panels used include those shown in figure B.8 through B.16.

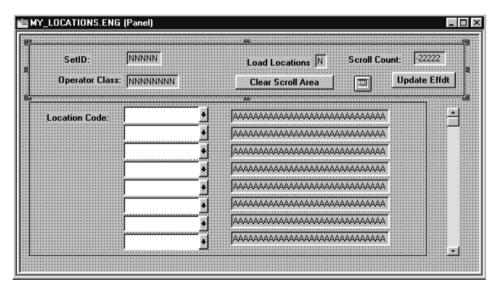


Figure B.8 MY\_LOCATIONS Panel

Drder Pa Num Lvl		Type	Field	Record
2 0 3 0 4 0 5 0	*** Top of List *** Frame SetID Operator Class Load Locations Clear Scroll Area Update Effect Update Effect Update Effect Scroll Count Frame Scroll Bar Location Code Dummy Name *** End of List ***	Frame Edit Box Edit Box Edit Box Push Button Push Button SecPanel Edit Box Frame Scroll Bar Edit Box Edit Box	SETID OPRCLASS LOAD_LOCATIONS MY_SCROLL_FLUSH EFFDT COUNTER LOCATION DESCR	MY_LOCATION_HDR MY_LOCATION_HDR MY_DERIVED MY_DERIVED MY_DERIVED MY_DERIVED MY_LOCATIONS LOCATION_TBL
	OK Car	ncel <u>S</u> elect	Move Unselect	Default

Figure B.9 MY\_LOCATIONS order of Panel

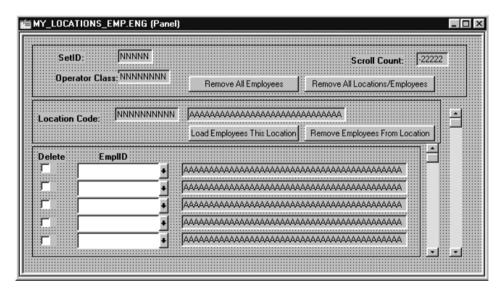


Figure B.10 MY\_LOCATIONS\_EMP Panel

Order Pa Num Lvl		Туре	Field	Record
1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 1 9 1 10 1 11 1 11 1 12 1 13 1 14 1	Frame Frame All Locations/Err Scroll Count Remove All Employees Operator Class SetID Frame Scroll Bar Location Code Durmy Name Load Employees This Loc Remove Employees From Frame Delete Scroll Bar	Frame Push Button Edit Box Push Button Edit Box Edit Box Frame Scroll Bar Edit Box Edit Box Push Button		MY_DERIVED MY_DERIVED MY_DERIVED MY_LOCATION_HDR MY_LOCATION_HDR MY_LOCATION_BL LOCATION_TBL MY_DERIVED MY_DERIVED
	Delete Row EmplID Name	Check Box Edit Box Edit Box	DELETE_ROW EMPLID NAME	MY_DERIVED MY_LOCATION_EMP IPERSONAL DATA
1 1012	OK Can		Move Unselect	Default

Figure B.11 MY\_LOCATIONS\_EMP Order of Panel

ť	MY_LOCATIONS.	GBL (Panel Group)				_ 🗆 )
	Panel Name	ltem Name	Hidden	ltem Label	Folder Tab Label	
1	MY_LOCATIONS	MY_LOCATIONS		My Locations		
2	MY_LOCATIONS_E	MY_LOCATIONS_EMP		My Locations Emp		1
2	MY_LOCATIONS_E	MY_LOCATIONS_EMP		My Locations Emp		

Figure B.12 Operator/Class & Employee Locations Panel Group

MY_L	LOCATIO	NS (I	Menu)		
<u>F</u> ile <u>E</u>	dit <u>V</u> iew	<u>G</u> o	Favorites	Locations Language	<u>H</u> elp
				Operator Locations	
				Operator/Employee Locations	

Figure B.13 Operator/Class & Employee Locations Menu

Menu Properties	X
General Use	
Menu Label: Operator Class Locations Menu Group	
&PeopleTools Applets &Compensate Employees 3 Define &Business Rules 9999 &Develop Workforce 1 &Administer Workforce 2 PeopleSoft 9999 &EDI Manager 9999 Monitot &Workplace 4 &PeopleTools Applets 5 Broblem Tracking 9999	
Menu Order     Menu Group O       Seguence #:     9393       ✓ Alphabetical order     □ Alphabetic       ✓ Add separator after     □ Add geparator	5 al <u>o</u> rder
	OK Cancel



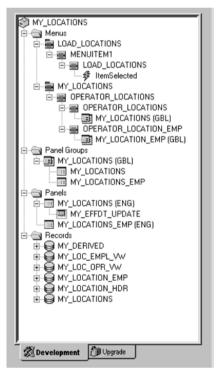


Figure B.15 Operator/Class Employee Location Project Workspace

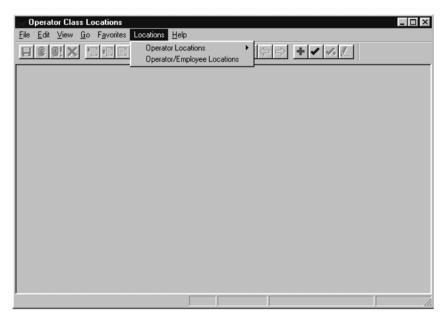


Figure B.16 Operator/Class & Employee Locations Applet as implemented

#### APPENDIX C

PeopleTool system tables

In this appendix you can find names and descriptions of the underlying PSTOOLS System tables. Some system tables have been omitted such as those that support Workflow and EDI Manager. The most common tables are listed by Tools Category.

#### **Application Engine**

AE_APPL_TBL	Application Definitions
AE_REQUEST	AE Request
AE_RUN_CONTROL	AE Run Control
AE_SECTION_TBL	Application Sections
AE_STEP_TBL	Section Steps
AE_STMT_B_TBL	AE Statement Chunk Table
AE_STMT_TBL	AE Statement Table

#### Change Control

PSCHGCTLHIST	Change Control History Table
PSCHGCTLLOCK	Change Control Locked Objects

Field Definition

PSDBFIELD

Database Field

#### **Record Definition**

PSDDLDEFPARMS

DDL Model Parameter

#### 1002

PSDDLMODEL PSIDXDDLPARM PSINDEXDEFN PSKEYDEFN PSRECDDLPARM PSRECDEFN PSRECFIELD PSSPCDDLPARM PSVIEWTEXT XLATTABLE DDL Model Statement Index DDL Parameters Index Definition Key in Index Definition Record Field PeopleCode Record DDL Parameter Record Definition Record Field Space DDL Parameters SQL View Text Translate Value

# **Panel Definition**

PSCOLORDEFN PSPNLDEFN PSPNLFIELD PSPNLTREECTRL PSSTYLEDEFN PSTOOLBARDEFN PSTOOLBARITEM Color Definition Panel Definition Panel Field Panel Tree Control Style Definition Toolbar Definition Toolbar Item

## Panel Group

PSPNLGROUP PSPNLGRPDEFN

## Menu Definition

PSMENUDEFNMenu DefinitionPSMENUITEMMenu ItemPSXFERITEMPop-up Menu Item Transfer Defns

Panel Group

Panel Group Definition

## **Operator Definition**

ACCESS\_GRP\_TBL PSACCESSPRFL PSAUTHITEM PSAUTHPRCS PSAUTHSIGNON PSOBJGROUP PSOPRALIAS PSOPRALIASTYPE PSOPRCLS PSOPRDEFN Tree Access Groups Access Profile Authorized Menu Item Authorized Process Authorized Signon Period Object Group Operator Alias Operator Alias Types Operator classes per operator Operator Definition

PEOPLETOOL SYSTEM TABLES

PSOPROBJ	Operator Object Group
PSPRCSPRFL	Process Profile
PSPRCSRUNCNTL	Process Run Control
SCRTY_ACC_GRP	Access Group Security
SCRTY_QUERY	PS/Query Profile

# **Tree Definition**

PSTREEDEFN	Tree Definition
PSTREELEAF	Tree Leaf
PSTREELEVEL	Tree Level
PSTREENODE	Tree Node
PSTREESTRCT	Tree Structure
TREE_LEVEL_TBL	Sample/Default Tree Level Tbl
TREE_NODE_TBL	Tree Nodes

# **Query Definition**

PSQRYBIND	Query Prompt
PSQRYCRITERIA	Query Criteria
PSQRYDEFN	Query Definition
PSQRYDEL	Query Definition
PSQRYEXPR	Query Expression
PSQRYFIELD	Query Field
PSQRYRECORD	Query Record
PSQRYSELECT	Query Select

## NVision Definition

NVS_REPORT	PS/n√
NVS_SCOPE	PS/n∖
NVS_SCOPE_FIELD	PS/n∖
NVS_SCOPE_VALUE	PS/n∖
PSTREESELCTL	Tree S
PSTREESELNUM	Tree S
PSTREESELECTxx	Tree S

# PS/nVision Report Requests PS/nVision Scope PS/nVision Scope Field PS/nVision Scope Values Tree Selection Control Tree Select Control Number Tree Select Work-Size (01 thru 30)

# **PeopleCode Definition**

PSPCMNAME PSPCMPROG PeopleCode Reference PeopleCode Program

# Utilities (Messages/Tablesets)

MESSAGE_CATALOG	Message Catalog
MESSAGE_SET_TBL	Message Sets

REC_GROUP_REC
REC_GROUP_TBL
SETID_TBL
SET_CNTRL_GROUP
SET_CNTRL_REC
SET_CNTRL_TBL
SET_CNTRL_TREE

Record Group Records Record Groups TableSet IDs TableSet Record Groups TableSet Record Detail TableSet Controls TableSet Tree Controls

Import Field

Import Definition

### Import Definitions

PSIMPFIELD PSIMPDEFN

Upgrader Definition

PSOBJCHNG PSPROJECTDEFN PSPROJECTITEM PSPROJECTMSG PSRELEASE PST\_PNLFIELDS

Object Change Project Definition Table Project Item Table Project Messages Release Table Upgrader Panel Work

#### **Process Scheduler**

PRCSDEFN PRCSDEFNGRP PRCSDEFNPNL PRCSDEFNXFER PRCSJOBDEFN PRCSJOBGRP PRCSJOBITEM PRCSJOBPNL PRCSRUNCNTL PRCSSAMPLER PRCSSYSTEM PRCSTYPEDEFN PSPRCSLOCK PSPRCSRQST **PSPRCSRQSTXFER PSRECURDEFN** PSSERVERSTAT SERVERCLASS SERVERDEFN

**Process Definitions** Process Definition Groups **Process Definition Panelgroups** Process Definition Transfers Process Job Definitions Process Job Groups Process Job Items Process Job Panel Groups Process Run Control Template Process Scheduler Example Process System Table Process Type Definitions Process Scheduler Lock Table Process Request Process Request Transfer Process Recurrence Definition **Process Server Statistics** Server Classes Process Server Definition

## **COBOL** Definition

MESSAGE\_LOGMessage Log TableMESSAGE\_LOGPARMMessage Parameter LogSQLSTMT\_TBLStored SQL Statements

## Mass Change

MC\_DATA\_TBL MC DEFN MC\_DEFN\_CRIT MC\_DEFN\_CRIT\_VL MC DEFN DEFAULT MC DEFN DESCR MC DEFN PT MC DEFN SQL MC DEFN SOL LN MC DEFN STMNT MC DTTM PARMS MC GROUP MC GROUP LN MC HIST CRIT MC HIST CRIT VL MC\_HIST\_DEFAULT MC HIST STMNT MC\_OPRID MC\_OPR\_SECURITY MC PROMPTS MC\_RUN\_CNTL MC\_TEMPLATE MC\_TEM\_CRITERIA MC\_TEM\_DEFAULTS MC\_TEM\_DESCR MC\_TEM\_STMNT MC\_TYPE MC\_TYPE\_DESCR MC TYPE FIELD MC\_TYPE\_JOIN MC\_TYPE\_RECORD MC TYPE SQL MC\_TYPE\_STMNT MC TYPE WHERE

Mass Change SQR Datatypes Mass Change Definition Mass Change Defn Criteria Mass Change Defn Crit Values Mass Change Defn Defaults Mass Change Defn Description Mass Change Defn PeopleTools Mass Change Defn SQL Mass Change Defn SOL Line Mass Change Defn Statement Mass Change Datetime Parms Mass Change Defn Group Mass Change Defn Group Line Mass Change History Criteria Mass Change History Crit Value Mass Change History Defaults Mass Change History Statement Mass Change Operator Security Mass Change Operator Security Mass Change Prompt Table Setup Mass Change Run Control Mass Change Template Mass Change Template Criteria Mass Change Template Defaults Mass Change Template Descr Mass Change Template Statement Mass Change Type Mass Change Type Description Mass Change Type Field Mass Change Type Join Table Mass Change Type Record Mass Change Type SQL Statement Mass Change Type Statement Mass Change Type Where Clause

# International Tables

COUNTRY\_TBL CURRENCY\_CD\_TBL Countries Currency Codes

## System Tables

PSASOFDATE PSCLOCK PSCOLORDEFN PSFMTDEFN PSFMTITEM PSLOCK PSOPTIONS PSSTYLEDEFN RUN\_CNTL\_SYSAUD STRINGS\_TBL SQR Request Dates Database Clock Access Color Definition Format Definition Table Format Item Table PeopleTools System Control PeopleTools System Options Style Definition SysAudit Control Table Strings Table

#### A P P E N D I X D

# Application Engine examples

In this appendix you can find the Application Engine source code used in our exercises. Each section/step is listed along with the statement type and the statement text used (if any). Any called sections appear next to the statement type (in the case of DO Select, DO When, or DO section types).

#### Exercise #1—Application USER001

Section/Step	MAIN.STEP1
Statement Type	Update
Statement Text	<pre>&amp;MSG(,1,'Hello World')</pre>

## Exercise #2—Application USER002

Section/Step	MAIN.STEP1		
Statement Type	Select		
Statement Text	&SELECT (COUNTER)		
	SELECT COUNT(*)		
	FROM PS_PERSONAL_DATA		
Section/Step	MAIN.STEP2		
Statement Type	Update		
Statement Text	&MSG(,2,'PERSONAL_DATA Record Count ',		
	&BIND (COUNTER))		

1008

Licensed to James M White <jwhite@maine.edu>

# Exercise #3—Application USER003

Section/Step	MAIN.STEP1
Statement Type	Select
Statement Text	&SELECT (COUNTER)
	SELECT COUNT(*)
	<pre>FROM PS_&amp;BIND(RECNAME,NOQUOTES,STATIC)</pre>
Section/Step	MAIN.STEP2
Statement Type	Update
Statement Text	&MSG(,3,
	&BIND(RECNAME, NOQUOTES),&BIND(COUNTER))

# Exercise #4—Application USER004

Section/Step	MAIN.STEP1
Statement Type	DO Select (Calls Section COUNT)
Statement Text	&SELECT (RECNAME)
	SELECT A.RECNAME
	FROM PSRECFIELD A,
	PSRECDEFN B
	WHERE A.RECNAME = B.RECNAME
	AND A.FIELDNAME = &BIND(FIELDNAME)
	AND B.RECTYPE = 0
	ORDER BY A.RECNAME
Section/Step	COUNT.STEP1
Statement Type	
Statement Text	&SELECT(COUNTER)
	SELECT COUNT(*)
	<pre>FROM PS_&amp;BIND(RECNAME,NOQUOTES,STATIC)</pre>
Section/Step	COUNT.STEP2
Statement Type	Update
Statement Text	&MSG(,3,
	&BIND(RECNAME,NOQUOTES),&BIND(COUNTER))

# Exercise #5—Application USER005

Section/Step	MAIN.ST	'EP1		
Statement Type	DO Sele	ect (Calls Section C	COU	NT)
Statement Text	&SELECT	(RECNAME)		
	SELECT	A.RECNAME		
	FROM	PSRECFIELD		A,
		PSRECDEFN		В
	WHERE	A.RECNAME	=	B.RECNAME

Licensed to James M White <jwhite@maine.edu>

	AND A.FIELDNAME = &BIND(FIELDNAME) AND B.RECTYPE = 0 ORDER BY A.RECNAME
Section/Step Statement Type	COUNT.STEP1
~ 1	&SELECT (COUNTER)
	SELECT COUNT(*)
	FROM PS_&BIND(RECNAME, NOQUOTES, STATIC)
Section/Step	COUNT.STEP2
Statement Type	Do When (Calls Section MSG)
Statement Text	&SELECT(AE_DECIDE)
	SELECT 'X'
	FROM PSLOCK
	WHERE &BIND(COUNTER) > 0
Section/Step	MSG.STEP1
Statement Type	Update
Statement Text	&MSG(,3,
	&BIND(RECNAME, NOQUOTES),&BIND(COUNTER))

# Exercise #6—Application USER006

Section/Step	MAIN.STEP1
Statement Type	DO Section (Calls Dynamic Section
	—HELLO or GOODBYE)
Statement Text	No Statement Text
Section/Step Statement Type Statement Text	HELLO.STEP1 Update &MSG(,1,'Hello World')
Section/Step Statement Type Statement Text	GOODBYE.STEP1 Update &MSG(,1,'Goodbye')

# Exercise #7—Application MYPRCSDL

I	MAIN.STE	P1		
Statement Type	Select			
Statement Text	&SELECT (	PRCSTYPE, PRCSNAME	)	
	SELECT B	B.PRCSTYPE, B.PRCSN	AM	Ε
	FROM F	PS_AE_RUN_CONTROL		A,
	]	PS_MY_RUN_CNTL_AE		В
	WHERE A	A.OPRID	=	B.OPRID

AND A.RUN CNTL ID = B.RUN CNTL ID AND A.PROCESS INSTANCE = &BIND (PROCESS INSTANCE) MAIN.STEP2 Section/Step Statement Type Update Statement Text &MSG(,2,&BIND(PRCSTYPE,NOQUOTES), &BIND (PRCSNAME, NOOUOTES)) Section/Step MAIN.STEP3 Statement Type DO Select (Calls Section DYNSECTN) Statement Text &SELECT (AE SECTION, RECNAME) SELECT 'PROCESS1, RECNAME FROM PSRECDEFN WHERE RECNAME = 'PRCSDEFN' OR RECNAME = 'PRCSDEFNGRP' OR RECNAME = 'PRCSDEFNPNL' OR RECNAME = 'PRCSDEFNXFER' UNION SELECT 'PROCESS2', RECNAME FROM PSRECDEFN WHERE RECNAME = 'PSPRCSROST' OR RECNAME = 'PSPNLFIELD' ORDER BY 1,2 Section/Step DYNSECTN.STEP1 Statement Type DO Section (Calls Dynamic Section —PROCESS1 or PROCESS2) Statement Text No Statement Text Section/Step PROCESS1.STEP1 Statement Type Update Statement Text &SELECT (COUNTER) SELECT COUNT(\*) FROM PS\_&BIND(RECNAME, NOQUOTES, STATIC) WHERE PRCSTYPE = &BIND(PRCSTYPE) AND PRCSNAME = &BIND(PRCSNAME) Section/Step PROCESS1.STEP2 Statement Type DO When (Calls Section DELETE1) Statement Text &SELECT (AE DECIDE) SELECT 'X' FROM PSLOCK WHERE &BIND(COUNTER) > 0

Section/Step Statement Type Statement Text	PROCESS1.STEP3 DO Section (Calls Section MESSAGE) No Statement Text	
Section/Step Statement Type Statement Text	DELETE1.STEP1 Update DELETE FROM PS_&BIND(RECNAME,NOQUOTES,STATIC) WHERE PRCSTYPE = &BIND(PRCSTYPE) AND PRCSNAME = &BIND(PRCSNAME)	
Section/Step Statement Type Statement Text	<pre>PROCESS2.STEP1 Update &amp;SELECT(COUNTER) SELECT COUNT(*) FROM &amp;BIND(RECNAME, NOQUOTES, STATIC) WHERE PRCSTYPE = &amp;BIND(PRCSTYPE) AND PRCSNAME = &amp;BIND(PRCSNAME)</pre>	
Section/Step Statement Type Statement Text	PROCESS2.STEP2 DO When (Calls Section DELETE2) &SELECT (AE_DECIDE) SELECT 'X' FROM PSLOCK WHERE &BIND(COUNTER) > 0	
Section/Step Statement Type Statement Text	PROCESS2.STEP3 DO Section (Calls Section MESSAGE) No Statement Text	
Section/Step Statement Type Statement Text	DELETE2.STEP1 Update DELETE FROM &BIND(RECNAME, NOQUOTES, STATIC) WHERE PRCSTYPE = &BIND(PRCSTYPE) AND PRCSNAME = &BIND(PRCSNAME)	
Section/Step Statement Type Statement Text	MESSAGE.STEP1 Update &MSG(,3,&BIND(RECNAME,NOQUOTES), &BIND(COUNTER))	

#### A P P E N D I X E

Built-in functions

# FREQUENTLY USED BUILT-IN FUNCTIONS

The following section lists frequently used built-in functions. Examples of these functions are also illustrated throughout the book.

PeopleCode built-in functions can be grouped into functional categories. Some of the more frequently used categories are:

- Conversion
- Date/Time
- Effective Date/Sequence
- Logical
- Math
- Message Catalog/Display
- Panel Buffer
- Panel Control
- Process Scheduler
- Save/Cancel
- Scroll functions
- SQL
- String
- Trace control
- Transfers

1013

# **Conversion functions**

## Char

*Description* Converts a numeric value to a character based on the character set in use

*Syntax* Char (n)

*Rules* Accepts one byte value only and not multiple values.

*Returns* Returns a string value based on the corresponding number passed to the function statement.

Example &CHAR\_VALUE = Char(70);

# Code

*Description* Examines the first character passed in a text string and returns the corresponding numeric code

Syntax Code(str)

*Rules* Double-byte characters are returned as numeric codes representing both bytes

Returns A number equal to the character set in use

```
Example
&NUMERIC_CODE = Code(&MY_STRING);
```

# ConvertChar

*Description* Converts the characters identified in the source string to the target character code

Syntax ConvertChar(source\_str, source\_str\_category, output\_str, target\_char\_code)

where:

source\_str identifies the source string to be converted source\_str\_category the language classification of the source string output\_str represents the converted string target\_char\_code numeric value identifying the conversion character type *Rules* Allows for conversion between character sets such as Japanese Hankaku Katakana, Zenkaku Katakana, and Hiragana. These character sets can also be converted to ASCII single-byte representation.

Source and target character code sets not supported by ConvertChar are processed by the function without alteration. The 0 and 1 characters are processed without conversion. A value of -1 is returned when ConvertChar cannot determine the placement of source and target characters. A -2 is returned when the characters in the source string can be partially converted. The characters that can be converted and the characters that cannot be converted are sent to the target string as they appear in the source string.

*Returns* A return code of 1 indicates the string was converted successfully and a 0 indicates the string was not converted. A -1 indicates an unknown condition.

```
Example
&RETURN_CODE = ConvertChar(&KANJI_STRING, 5, ASCII_STRING, 0);
```

# String

Description Takes a value stored in a non-string type and converts it to a string

Syntax String(val)

*Rules* String can be used when field comparisons require the specific use of string data. Object data types cannot be converted using the String function.

Returns A string representation of the value passed to the function

```
Example
&MY_STRING = String(&NUMBER_FIELD);
```

# **Date/Time functions**

## Date

*Description* Converts a number formatted as YYYYMMDD and returns a DATE value

Syntax Date (date\_number)

*Rules* The input format must be a number.

*Returns* A DATE data type value

Example
&DATE\_FIELD = Date(20000101);

### DatePart

Description Returns part of an input DateTime value

Syntax DatePart(datetime\_value)

*Rules* The input value is a DateTime data type.

Returns Returns the date value portion

```
Example
&DATE_PORTION = DateTimeValue("01/01/00 06:30:25 AM");
&DATE_PORTION = DatePart(&DATE_PORTION);
```

## DateValue

*Description* Converts a date string that is in the Windows standard date setting and returns a date type

Syntax DateValue(date\_str)

*Rules* When the input date value is in the YY/MM/DD format and Windows regional date setting is MM/DD/YY, the panel processor returns an invalid date function error message.

Returns Returns values based on the Windows regional date setting

#### Example

```
/* Date format is mm/dd/yy */
&DATE_FIELD = DateValue("01/01/00");
/* Date format is yy/mm/dd */
&DATE_FIELD = DateValue("00/12/31");
```

# AddToDate

*Description* Accepts a date and three additional parameters representing number of years, months, and days. The specified values are added to the date.

Syntax
AddToDate (date, num\_years, num\_months, num\_days)

*Rules* The function accounts for leap years and will subtract from the specified date when negative values are passed.

*Returns* A date representing the date passed and +/- the number of years, months and days.

/\* Subtracts five years from current date, then adds six months and two days  $^{\star/}$ 

#### Example

```
&CALCULATED_DATE = AddToDate(%Date, -5, 6, 2);
```

# **Effective Date/Sequence functions**

### CurrEffdt

*Description* Identifies and returns a value representing the current effective date for the record at the current scroll level

Syntax CurrEffDt(level\_number)

*Rules* When a level is not specified, the effective date of the current scroll level is returned.

*Returns* The return value is a Date type. When used in a scroll area that does not contain an effective dated record in the primary scroll, the function returns a number.

```
Example
&EFFDT = CurrEffDt(2);
```

# CurrEffSeq

Description Returns the effective sequence of a specified scroll area

Syntax CurrEffSeq(level\_num)

*Rules* When a level is not specified, the effective sequence returned is that of the current scroll level.

Returns A number data type representing the effective sequence of the scroll area

### Example

```
If CurrEffSeq(CurrentLevelNumber()) <> &PREVIOUS_EFFSEQ Then
   SetDefault(JOB.ACTION);
   SetDefault(JOB.ACTION_REASON);
End-If;
```

## CurrEffRowNum

Description Returns the effective row number of the current specified scroll area

```
Syntax
CurrEffRowNum(level_number)
```

*Rules* When a level is not specified, the effective row number returned is that of the current scroll level.

*Returns* A number data type representing the effective row number of the scroll area

*Example* &ROW\_NUMBER = CurreffRowNum(2);

### NextEffDt

*Description* Returns the value of the specified record field that exists in the next effective-dated row

Syntax NextEffDt(record\_field)

*Rules* Works only with effective dated records. When a next record does not exist, the statement is bypassed.

Returns An Any data type containing the field in the next effective-dated record

#### Example

```
If JOB.DEPTID = NextEffdt(JOB.DEPTID) Then
    Gray(JOB.PAYGROUP);
End-If;
```

# PriorEffDt

*Description* Works with effective-dated records and is a contrast to NextEffDt. It returns the contents of the field passed to the function statement that appear in the prior effective-dated row

Syntax PriorEffdt(record\_field)

*Rules* Works only with effective dated records. When there is no prior record, the statement is bypassed by the Application Processor.

*Returns* An Any data type containing the field from the prior effective-dated record

#### Example

```
If JOB.COMPRATE < PriorEffdt(JOB.COMPRATE) Then
    Error ("New Compensation Rate cannot be < previous rate");
End-If;</pre>
```

# **Logic functions**

# **All**

*Description* Determines whether one or more fields contain a value. The All function is useful in the SaveEdit PeopleCode event if we wish to verify that one or more fields have been entered.

Syntax All(fieldlist)

Rules fieldlist represents one or more field names. They can be specified as [recordname.] fieldname1 [, [recordname.] fieldname2]

*Returns* A Boolean. If all the fields contain a value, then the function returns True. If one or more fields do not contain a value, the function returns False.

### Example

```
If All(MY_PROBLEM_RESOLTN, CLOSE_DT) Then
   Gray(MY_PROBLEM_STATUS);
End-If;
```

# AllOrNone

*Description* This function is a combination of the All and None functions. The function returns a Boolean True when all the fields contain values or if none of the fields contain values. False is returned when there is a combination of fields containing values and fields that do not contain values.

Syntax AllOrNone(FieldList)

*Rules* A character field containing blanks or a numeric field containing a zero is categorized as a null field value.

Returns True when all fields contain values or none of the fields contain values.

### Example

```
If AllOrNone(CLOSE_DT, MY_PROBLEM_RESOLTN) Then
    &RETURN = MyAuditFunction();
End-If;
```

## None

Description Verifies a character field contains blanks or a numeric field contains zero

Syntax None(FieldList) *Rules* A character field containing blanks or a numeric field containing a zero is categorized as a null field value.

*Returns* Returns True if the field or list of fields do not contain a value. A False is returned if one or more fields contain a value.

#### Example

```
If MY_PROBLEM_STATUS = "5" Then
    If None(CLOSE_DT) Then
        Error ("Close date is required for resolved issues");
    End-If;
End-If:
```

# Math functions

### Round

*Description* Rounds up the number passed to the specified number of decimal positions.

Syntax Round (decimal, precision)

*Rules* The value represented by a decimal must be of a number data type.

*Returns* Returns a decimal number rounded up to the number of decimal positions in precision.

Example
&ANNUAL\_RT = Round(JOB.COMPRATE, 3);

# Int

*Description* Removes the decimal positions from a number and returns an integer value.

Syntax Int(decimal\_number)

*Rules* Int does not round the input value; it only truncates the decimal positions.

Returns Returns a whole number value.

# Example

```
&HOURLY_RATE = 12.675;
&NEW_RATE = Int(&HOURLY_RATE);
/* Value of &New_rate is now 12 */
```

### Truncate

Description Removes the specified number of digits from a decimal value.

Syntax Truncate (decimal\_number, digits)

*Rules* Does not perform any rounding. When the parameter identified by digits contains a zero, all numbers to the right of the decimal point are removed.

*Returns* Returns a number value.

#### Example

```
&COMPENSATION_RATE = 2375.67895;
&NEW_RATE = Truncate(&COMPENSATION_RATE, 3);
/* Value of &New_Rate is 2375.678 */
   &COMPENSATION_RATE = 2375.67895;
   &NEW_RATE = Truncate(&COMPENSATION_RATE, 0);
/* Value of &New_Rate is 2375 */
```

# **Message Catalog/Display functions**

### MessageBox

Description Creates and displays a message box window.

#### Syntax

```
MessageBox(style, title, message_set, message_num,
default_txt [, paramlist])
```

#### where

style Enables the message box window to be tailored with a blend of icons and push buttons.

title The message box title.

message\_set message\_set of the message catalog. Message sets 1 through 19,999 are reserved for PeopleSoft applications.

message\_num The message number within the message set.

default\_txt Text that is displayed in the message box when the cataloged message set is not available or message set is represented by zero.

parmlist List of parameters that are displayed in the text string. They can be represented as %1, %2 and so on.

*Rules* The function return value can be interpreted if necessary. With the style parameter, two or more buttons can be included in the message box, but their use is limited to certain PeopleCode events. When the style parameter is left out or style contains more than one button, the function becomes user think-time, which

indicates the button action returns a value back to the function. As a result of awaiting a reply, the Application Processor suspends the PeopleCode program until the user clicks on one of the buttons contained in the message.

*Returns* A number value indicating which button was pressed. See table E.1 for a list of return value descriptions. A return value of zero indicates there was insufficient memory to construct the MessageBox.

#### Example

```
If MY_PROBLEM_STATUS = "2" Then
    MessageBox(289, "Incorrect Data", 20012, 1, "Project ID %1 is invalid",
MY_PROJECT_ID);
End-If;
```

Returns	Description
0	Insufficient memory
-1	Warning
1	OK button was pressed
2	Cancel
3	Abort
4	Retry
5	Ignore
6	Yes
7	No

Table E.1 Message return values

# WinMessage

*Description* WinMessage is used to display information in a message box.

## Syntax

```
WinMessage (message [, style] [, title]
```

message A text string displayed in the message box. When WinMessage is used as a debugging tool, a text string provides valuable information by including field contents as parameters. Utilizing WinMessage to assist while debugging does not require the use of MsgGet and MsgGetText functions.

- style This parameter is optional.
- title The message box title.

*Rules* From a debugging perspective, WinMessage can be used to display field contents and allow us to "inch" through PeopleCode statements if necessary.

*Returns* When the style parameter is passed, WinMessage returns a number indicating which button was pressed. See table E.1 for a list of return value descriptions.

A return value of zero indicates there is insufficient memory to construct the message box.

When the style parameter is not included, a Boolean value is returned. True indicates the OK button is pressed.

#### Example

/\* This example does not use style or title \*/
WinMessage("A message with no style!");

/\* This message has style and returns a value based on button pressed \*/
WinMessage("Close date cannot be less than the reported incident date", 289,
"Invalid Date");

## Error

*Description* Is used to display an error message and stop processing of the active panel. Error works with messages stored in the Message Catalog or with a text string supplied to the function.

Syntax Error (String)

*Rules* The value contained in string can be a literal text message or a message stored in the message catalog. The stored message must be retrieved using the MsgGet or MsgGetText functions. This is important when using translated text messages. Error terminates the PeopleCode program and prevents further statements from being executed. Error, however, produces varying results from one PeopleCode event to another. The events in which Error is commonly used include FieldEdit and SaveEdit. When executed in these events, the message is displayed and processing is halted. In FieldEdit, the field that contains the PeopleCode event is highlighted; in SaveEdit, no fields are highlighted. One manner in which to work around this in the SaveEdit event is to use the SetCursorPos function for the field, prior to calling Error. RowDelete is another PeopleCode event in which the Error function is sometimes used. When Error is called in RowDelete, the message is displayed and the row is not deleted.

#### Returns Does not return a value

### Example

```
/* Implemented with a message string */
Error ("All field values are required");
/* Used with a cataloged message */
Error MsgGet(20010, 1, "All field values are required");
```

## Warning

*Description* Warning is used to display a message. Warning differs from Error because it does not halt processing. The user is presented with OK and Explain buttons, then has the opportunity to correct or change data.

Syntax Warning (String)

*Rules* Warning works with messages stored in the message catalog or a text string supplied to the function. The stored message must be retrieved using the MsgGet or MsgGetText function. When executed, the Warning statement terminates the PeopleCode program and prevents further statements from being executed. Warning produces varying results from one PeopleCode event to another. The events in which Warning is commonly used include FieldEdit and SaveEdit. When used in FieldEdit, the message is displayed and the field that contains the PeopleCode is highlighted. Placing Warning in SaveEdit displays the message but does not highlight fields. One manner in which to work around this in the SaveEdit event is to use the SetCursorPos function for the field prior to Warning. RowDelete is another PeopleCode event in which Warning is sometimes used. When Warning is called in RowDelete the message is displayed with OK and Cancel buttons. The user then has the option to delete the row by pressing OK or to back out of the delete by pressing Cancel.

#### Returns Does not return a value

### Example

/\* This message enables the user to continue after pressing OK \*/

Warning("Incident status has been assigned");

# MsgGet

*Description* MsgGet retrieve messages from the message catalog and, when necessary, substitutes the value of each parameter contained in the message text identified by %1, %2, %3.

#### Syntax

MsgGet (message\_set, message\_num, default\_msg\_text [, paramlist] )

*Rules* When a message set number less than 1 is passed, or if the message is not in the catalog, the default message text is substituted.

MsgGet is not a separate function. It is used in conjunction with MessageBox, WinMessage, Error, and Warning.

*Returns* Retrieves stored message and substitutes parameter in a paramlist, but does not return a value

```
Example
```

### MsgGetText

Description MsgGetText retrieve messages from the Message Catalog and when necessary substitutes the value of each parameter contained in the message text identified by %1, %2, %3.

Syntax MsgGetText (message\_set, message\_num, default\_msg\_text [, paramlist] )

*Rules* MsgGetText is almost identical to MsgGet except that the function displays the message without displaying a message set and message number.

*Returns* Retrieves stored message and substitutes parameter in a paramlist, but does not return a value

```
Example
Error (MsgGetText(20012, 1, "Data cannot be saved until all fields are
entered"));
```

# Panel Buffer functions

# DeleteRecord

*Description* Works on a level zero scroll record and is used to remove the parent and any corresponding child records from the database

```
Syntax DeleteRecord (level_zero_recfield)
```

*Rules* Marks records to be deleted. During save processing, the row is deleted from the database. The DeleteRecord function cannot be executed from a Save-PostChange or WorkFlow PeopleCode event because database updates are performed at the conclusion of the Workflow event.

*Returns* An optional Boolean value is returned following the completion of the function

```
Example
&RETURN_VALUE = DeleteRecord(MY_PROJECT_ID);
```

# FieldChanged

*Description* Is used to verify if one or more specified fields have been changed. A field can be changed on a panel or by a PeopleCode program.

*Syntax* There are two methods of implementing FieldChanged, and they are based on how the field is referenced. When the field is referenced in a scroll path, the syntax is

```
FieldChanged(scrollpath, target_row,
    [recordname.] fieldname)
```

When the field is referenced by context: FieldChanged( [recordname.] fieldname)

*Rules* When performed from a record definition that is not the same as the record name, then the record name prefix is required.

*Returns* Returns True when the contents of the Record.Fieldname have been changed since being retrieved from the database

```
Example
```

```
If FieldChanged(PRIORITY) Then
    &RETURN = MyAuditFunction();
    End-If;
```

# InsertRow

*Description* Inserts a new row of data into the scroll buffer. The operation is followed by the RowInsert PeopleCode event.

Syntax

InsertRow (scrollpath, target\_row [, turbo])

*Rules* This function performs the same steps as if the F7 key were pressed. The InsertRow function is immediately followed by the RowInsert PeopleCode event. The remaining PeopleCode events then follow RowInsert. For effective-dated scrolls, the new row is inserted before the target row. When a non-effective-dated record is inserted, the new row is inserted after the row identified in the function. For effective-dated rows, the Effdt field is set to the current date, and the values that exist in the previous row are copied to the newly inserted row.

*Returns* An optional Boolean value is returned following the completion of the function.

Example
InsertRow(RECORD.MY\_LOCATIONS,
CurrentRowNumber(), RECORD.MY\_LOCATION\_EMP);

# PriorValue

Description Returns the prior value of a buffer field

Syntax PriorValue(fieldname)

*Rules* To expect correct results, this function should be used in the FieldEdit and FieldChange events for the buffer field where PriorValue is called. When the value of a field is '1' during panel startup, and the value is then changed by the user to a '2' and then to '3', the PriorValue function returns '2' when executed after the second change. The value will not be the initial '1'.

Returns Returns an Any data type

```
Example
    If PriorValue(DESCRLONG) = " " Then
        CLOSE_DT = %Date;
        End-If;
```

## RecordChanged

*Description* Indicates whether a row has been modified since being retrieved from the database

Syntax Contexual Reference: RecordChanged(RECORD.target\_recname)

When the PeopleCode program executing is on the same record, we can use RecordChanged(recordname.fieldname)

*Rules* Can be used during save processing to identify updates based on changes made during a panel session

*Returns* Returns True if the record was changed by a user panel or changed from within a PeopleCode program

```
Example
```

```
If RecordChanged(MY_USER_TABLE.NAME) Then
    &RETURN = MyAuditFunction();
End-If;
```

## RecordDeleted

*Description* Can be used to identify rows marked for deletion as a result of an operator F8 delete or a program DeleteRow function call

*Syntax* There are two methods of implementing the RecordDeleted function, and they are based on how the row is referenced. When the row is referenced in a scroll path, the syntax is

```
RecordDeleted(scrollpath, target_row)
```

When the row is referenced by context RecordDeleted (RECORD.target\_recordname)

*Rules* Deleted rows are removed from the buffer during save processing, which enables the RecordDeleted function to be used in most events up to and including SavePostChg.

Returns Returns a Boolean True when a row has been marked for deletion

### Example

```
If RecordDeleted(RECORD.MY_LOCATIONS,
CurrentRowNumber(), RECORD.MY_LOCATIONS_EMP) Then
MY_DERIVED.COUNTER = ActiveRowCount(RECORD.MY_LOCATIONS,
CurrentRowNumber(), RECORD.MY_LOCATION_EMP);
End-If;
```

# RecordNew

*Description* Used during save processing to determine if a row is new to the database

*Syntax* Can be used in two ways based on how the row is referenced. When the row is referenced in a scroll path, the syntax is

```
RecordNew(scrollpath, target_row)
```

```
When the row is referenced by context
RecordNew(RECORD.target_recordname)
```

*Rules* In previous releases of PeopleCode this could be written as Record-New(Recordname.Fieldname)

*Returns* Returns a Boolean True when the record is new to the panel buffer.

## Example

```
/* Using scrollpath */
If RecordNew(RECORD.MY_LOCATIONS, CurrentRowNumber(),
RECORD.MY_LOCATION_EMP) Then
    &RETURN = MyAuditFunction();
End-If;
/* Using contextual reference) */
If RecordNew(RECORD.MY_LOCATIONS) Then
    &RETURN = MyAuditFunction();
End-If;
```

# **Panel Control functions**

## Gray

Description Sets a field on a panel so that it is display only and cannot be changed

Syntax Gray(fieldname)

*Rules* The Gray function is commonly used in the RowInit event and can appear in other events such as FieldChange.

*Returns* Returns a Boolean that can be used to determine if the function was successful

```
Example
```

```
If MY_PROBLEM_STATUS = "5" Then
Gray(CLOSE_DT);
End-If;
```

# Hide

Description Hides a field on a panel making it invisible to the user

Syntax Hide (fieldname)

*Rules* Hide can be used in a RowInit event, but can also appear in events such as FieldChange when fields are hidden based on changes made to corresponding data elements.

*Returns* Returns a Boolean, which can be used to determine if the function was successful

```
Example
If MY_PROBLEM_STATUS <> "5" Then
Hide(MY_PROBLEM_RESOLTN);
End-If;
```

# UnHide

Description UnHide makes a panel field visible again.

Syntax UnHide(fieldname)

*Rules* Fields that are hidden based on the Panel Field Properties-Use-Invisible tab are not made visible because UnHide has no impact on these fields.

*Returns* Returns a Boolean, which can be used to determine if the function was successful

### Example

```
If PERSONAL_DATA.BIRTHCOUNTRY <> "USA" Then
    UnHide(PERSONAL_DATA.BIRTHSTATE);
End-If;
```

## Ungray

Description Allows a previously non-editable field to be editable

Syntax Ungray (fieldname)

*Rules* Used in events such as RowInit. Can also appear in FieldChange after the status of a field is impacted by changes to its value or other corresponding fields.

*Returns* Returns a Boolean, which can be used to determine if the function was successful

```
Example
If COMPANY = DEPT_TBL.COMPANY Then
Hide(COMPANY, PAYGROUP);
End-If;
```

# Process Scheduler functions

## ScheduleProcess

*Description* The ScheduleProcess function stores a row in the Process Request table enabling the system to schedule a process or job.

#### Syntax

```
ScheduleProcess(process_type, process_name
[, run_location] [, run_cntl_id] [, process_instance]
[, run_dttm] [, recurrence_name] [, server_name])
```

where

process\_type A case-sensitive string that identifies the type of process to be run. SQR Report and Application Engine are examples of process types.

process\_name An eight-character string used to identify the process

run\_location A one character string that identifies if the process is run on the client ('1') or the server ('2')

run\_cntl\_id Identifies the Run Control ID that links operator IDs to Run Controls

process\_instance The ScheduleProcess function receives this as a variable and assigns a unique number to identify each process requested.

run\_dttm A process or job can be scheduled for some future time by passing a DateTime value in this parameter. The %DateTime system variable can also be passed for immediate scheduling.

recurrence\_name Identifies the name of a recurring job or process

server\_name Identifies the server on which the process or job will be run

*Rules* process\_type and process\_name are the only required parameters necessary to schedule a process.

When a call to ScheduleProcess is made from a program running on an application server, the run\_location parameter cannot be passed as '1' (client). Doing so generates an error and subsequent cancellation of the request.

Any process involving COBOL or SQR scheduled from a program on an application server must also be run on the server. When the PeopleCode program containing ScheduleProcess is run on a client, the COBOL or SQR process is not restricted and can run on either the client or server.

The parameter list can accept strings in the form of bind variables or a Meta-SQL string.

*Returns* A successful process returns zero. A non-zero return indicates an error was encountered.

### Example

```
If ScheduleProcess("SQR Report", &REPORT_NAME,
    &RUN_LOCATION, &RUN_CNTL_ID) = 0 Then
    WinMessage("SQR Report successfully scheduled");
End-If;
```

# **Save/Cancel functions**

## DoCancel

*Description* Used to cancel activity on a panel. The function mimics the ESC key and the Cancel toolbar icon.

Syntax DoCancel()

*Rules* For the current panel group, all PeopleCode programs are terminated except for those executing in the following events:

```
SaveEdit
SavePreChg
SavePostChg
```

Returns Does not return a value

```
Example
If &RETURN_CODE <> 0 Then
    DoCancel();
End-If;
```

# DoSave

*Description* Performs save processing at the conclusion of the current PeopleCode program in the FieldEdit, FieldChange, and MenuItemSelect events.

*Syntax* DoSave ()

*Rules* PeopleCode programs containing DoSave continue processing until the remaining statements are executed. The panel is saved at the conclusion of the program. Save processing includes the following events:

SaveEdit SavePreChg SavePostChg WorkFlow

Returns Does not return a value

```
Example
If &RETURN_CODE = 0 Then
    DoSave();
End-If;
```

## DoSaveNow

*Description* Works similar to DoSave, however, the panel is immediately saved without waiting for the PeopleCode program to conclude.

Syntax DoSaveNow ( )

*Rules* After the panel is saved, any remaining PeopleCode statements that follow DoSaveNow are executed.

DoSaveNow is only valid from the FieldEdit and FieldChange events.

A common use of DoSaveNow is when remote calls are involved. When using RemoteCall, DoSaveNow can be used to save information to the database before a remote process is called.

Returns Does not return a value

```
Example
```

## WinEscape

Description Used to cancel activity on a panel. WinEscape mimics the ESC key.

Syntax WinEscape ( )

*Rules* Changes made to the panel since the previous save are revoked

Returns An optional Boolean value is returned if required.

#### Example

```
/* This example cancels the panel when fields are missing */
If None(MY_PROBLEM_STATUS, PRIORITY, MY_USER_ID) Then
   WinEscape();
End-If;
```

# Scroll functions

# ScrollSelect

*Description* Selects records from a table and loads them into the scroll buffer area of a panel. Inserts child rows under the next higher level row.

#### Syntax

```
ScrollSelect (levelnum, [RECORD.level1_recname,
[RECORD.level2_recname,]] RECORD.target_recname,
RECORD.sel_recname
[, sqlstr [, bindvars]]
[, turbo])
```

#### where

levelnum The level number of the target scroll area. This value can be 1, 2, or 3.

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede target\_ recordname.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by RECORD.level1\_recname and RECORD.level2\_recname.

RECORD.target recordname The target scroll area where the selected data are loaded. When the target scroll is on level 3, specify the level 1 and level 2 records first followed by the level 3 target scroll.

RECORD.sel\_recordname Specifies the record or view to retrieve data from. The sel\_recordname can be the same as target\_recordname. One characteristic of this parameter is that it enables target rows to be loaded into a buffer with only those fields used in the scroll area, in addition to key fields. When selecting rows from a large table such as JOB (in HRMS) and the target scroll area only uses five fields, specifying a smaller target reduces the amount of data loaded into system buffers.

sqlstr [, bindvars] The optional SQL string parameter can contain an SQL WHERE and ORDER BY clause. One or both can be specified. The WHERE clause enables us to limit the number of rows loaded into the scroll area. The ORDER BY clause can be used to sort the rows before being loaded into the target scroll area. The SQL string can accept bind variables that are used as part of the WHERE or ORDER BY clause. Bind variables can be regular bind or inline bind variables. SQL string can include Meta-SQL functions.

turbo When specified, improves performance of the ScrollSelect function. The parameter is passed as a Boolean True.

*Rules* Allows for the specification of the target scroll area, a source record from which to select rows, and an optional SQL string. Keys on the select record must be the same as on the target scroll record. A record can be used as both select and scroll record. Select record must be defined and created using Application Designer. Select record cannot be a Derived/Work record.

Returns Does not return a value

#### Example

## ScrollSelectNew

*Description* Resembles ScrollSelect, except that ScrollSelectNew marks records as new when they are loaded into the scroll area. During save processing, these records are automatically added to the database.

```
Syntax
ScrollSelectNew (levelnum,
[RECORD.level1_recname, [RECORD.level2_recname,]]
RECORD.target_recname, RECORD.sel_recname
[, sqlstr [, bindvars]]
[, turbo])
```

where

levelnum Represents the level number of the target scroll area. This value can be 1, 2, or 3.

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by RECORD.level1\_recname and RECORD.level2\_recname.

RECORD.target recordname The target scroll area where the selected data are loaded. When the target scroll is on level 3, we need to specify the level 1 and level 2 records first, followed by the record at target scroll level 3.

RECORD.sel\_recordname Specifies the record or view to retrieve data from. The sel\_recordname can be the same as target\_recordname. One characteristic of this parameter is that it allows target rows to be loaded into a record with only those fields used in the scroll area in addition to key fields.

sqlstr [, bindvars] The optional SQL string parameter can contain an SQL WHERE and ORDER BY clause. One or both can be specified. The WHERE clause enables us to limit the number of rows loaded into the scroll area. The ORDER BY clause can be used to sort the rows before they are loaded into the target scroll area. The SQL string can accept bind variables that are used as part of the WHERE or ORDER BY clause. Bind variables can be regular or inline bind variables. The SQL string can include Meta-SQL functions.

turbo Improves performance of the ScrollSelectNew function. The parameter is passed as a Boolean True when Turbo is used.

*Rules* Keys on the select record must be the same as on the target scroll record. A record can be used as both select and scroll record. Select record must have been defined and created using Application Designer and cannot be a Derived/Work record.

#### Returns Does not return a value

#### Example

### ScrollFlush

Description Removes records from a target scroll area

```
Syntax
ScrollFlush (scrollpath)
ScrollPath defined as
[RECORD.level1_recname, level1_row,]
[RECORD.level2_recname,] level2_row,]
RECORD.target_recname
```

where

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname. The leve1\_recname requires the RECORD prefix.

level1\_row The level 1 row to flush. The value is an integer and can be a variable or a constant. The parameter must be specified when ScrollFlush is targeted at scroll levels 2 or 3.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row Indicates the scroll level 2 to flush. The value is an integer and can be a variable or a constant. The parameter must be specified when Scroll-Flush is targeted at scroll level 3.

Target recordname The target scroll area where rows to remove are located. When the target scroll is on level 3, specify the level 1 and level 2 records first, then the record at target level 3. The target record name must be prefixed by RECORD.

*Rules* Rows flushed from the target scroll area are not removed from the database

Returns Does not return a value

#### Example

```
ScrollFlush(RECORD.MY_LOCATIONS, CurrentRowNumber(),
RECORD.MY_LOCATION_EMP, RECORD.MY_LOC_EMPL_VW);
```

# ActiveRowCount

Description Identifies the sum of active rows in a given scroll area

### Syntax

```
ActiveRowCount (Scrollpath)
ScrollPath defined as:
[RECORD.level1_recname, level1_row,]
[RECORD.level2_recname,] level2_row,]
RECORD.target_recname
```

where

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2 this parameter must precede the target\_recordname. The leve1\_recname requires the RECORD prefix.

level1\_row Identifies the record at scroll level 1. The value is an integer and can be a variable or a constant. The parameter must be specified when ActiveRowCount is used to return the number of active rows at scroll level 2 or 3.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row Identifies the record at scroll level 2. The value is an integer and can be a variable or a constant. The parameter must be specified when ActiveRowCount is used to return the number of active rows at scroll level 3.

Target recordname Record in the target scroll area. The target record name must be prefixed by RECORD. The target record may be on scroll level 1, 2, or 3

Rules Records marked as deleted are not included in the count.

Returns Returns a number representing the number of active rows in a scroll area

#### Example

```
&NUMBER_OF_ROWS = ActiveRowCount(RECORD.MY_LOCATIONS,
CurrentRowNumber(), RECORD.MY_LOCATION_EMP);
```

# CurrentRowNumber

*Description* CurrentRowNumber is used when it is necessary to identify the row number of the current row in a scroll area.

Syntax CurrentRowNumber ( [level] )

**Of** CurrentRowNumber()

where

level Identifies the scroll level where the row number is retrieved

*Rules* When the level parameter is not specified, the function uses the current scroll level from where the function is called as the default level. CurrentRowNumber is sometimes used with ActiveRowCount to limit program loops to the number of active rows.

*Returns* A number representing the current row number on the specified scroll level

### Example

```
/*The return value can be used in ActiveRowCount*/
   &COUNT = ActiveRowCount(RECORD.MY_LOCATIONS, &ROW_NUMBER,
   RECORD.MY_LOCATION_EMP);
```

```
/*CurrentRowNumber can also be specified explicitlty */
   &COUNT = ActiveRowCount(RECORD.MY_LOCATIONS,
   CurrentRowNumber(1), RECORD.MY_LOCATION_EMP);
```

# DeleteRow

*Description* DeleteRow enables rows to be deleted from a PeopleCode program. The function triggers the RowDelete event, which mimics the F8/Delete Row operation.

#### Syntax

```
DeleteRow (Scrollpath, target_row )
```

### ScrollPath defined as:

```
[RECORD.level1_recname, level1_row, ]
[RECORD.level2_recname,] level2_row, ]
RECORD.target_recname
```

#### where

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname. The leve1\_recname requires the RECORD prefix.

level1\_row Identifies the record at scroll level 1. The value is an integer and can be a variable or a constant. The parameter must be specified when DeleteRow is used to delete rows at scroll level 2 or 3.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row Identifies the record at scroll level 2. The value is an integer and can be a variable or a constant. The parameter must be specified when DeleteRow is used to delete the number of rows at scroll level 3.

Target recordname The target scroll area to delete. The target record may be on scroll level 1, 2, or 3 and must be prefixed by RECORD.

target\_row Identifies the row number to be deleted

*Rules* When DeleteRow is used in a loop, the operation must begin with the highest row and work downwards. Each time a row is deleted the system renumbers all remaining rows.

Returns Returns an optional Boolean value

#### Example

```
For &I = ActiveRowCount(RECORD.PERS_DATA_EFFDT)
To 1 Step - 1
DeleteRow(RECORD.PERS_DATA_EFFDT, &I);
End-For;
```

## FetchValue

*Description* Retrieves the value of a field from a row stored in the panel buffer of a scroll area and places it into a variable or fieldname

#### Syntax

```
FetchValue (Scrollpath, target_row,
[recordname.] fieldname )
```

#### ScrollPath is defined as:

```
RECORD.level1_recname, level1_row, ]
[RECORD.level2_recname,] level2_row, ]
RECORD.target_recname
```

#### where

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname. The level\_recname requires the RECORD prefix.

level1\_row Indicates the scroll level 1 row. The value is an integer and can be a variable or a constant. The parameter must be specified when fields from rows at level 2 or 3 are fetched.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row Represents the scroll level 2 row to be referenced. The value is an integer and can be a variable or a constant. The parameter must be specified when FetchValue is targeted at scroll level 3.

Target recordname Represents the target scroll area containing the row where data are to be fetched from. The target record name must be prefixed by RECORD. The target record may be on scroll level 1, 2. or 3.

target\_row Identifies the row number in the target scroll area where buffer field contents we will be retrieved.

[recordname.] fieldname The name of the field that references the value to be loaded. The record name is used when the function call is made from a record definition that is not the same as recordname. The fieldname can reside on scroll level 1, 2, or 3.

*Rules* In many instances FetchValue may not be necessary if the contents of a field are accessible to a program by using the [recordname].fieldname syntax. FetchValue can be used when a value is not within context.

*Returns* Returns an ANY data type value.

```
Example
&EMPLID = FetchValue(RECORD.MY_LOCATIONS,
CurrentRowNumber(), RECORD.MY_LOCATION_EMP, &I,
MY_LOCATION_EMP.EMPLID);
```

# HideRow

*Description* HideRow is used to hide a specific row and any child rows in subordinate scroll levels.

```
Syntax
HideRow (Scrollpath)
```

[, target\_row]

### ScrollPath defined as:

```
[RECORD.level1_recname, level1_row, ]
[RECORD.level2_recname,] level2_row, ]
RECORD.target_recname
```

#### where

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname. The leve1\_recname requires the RECORD prefix.

level1\_row This parameter indicates the scroll level 1 row. The value is an integer and can be a variable or a constant. The parameter must be specified when fields from rows at level 2 or 3 are to be hidden.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row This parameter represents data at scroll level 2. The value is an integer and can be a variable or a constant. The parameter must be specified when HideRow is targeted at scroll level 3.

Target recordname The target record to hide. The target record may be on scroll level 1, 2, or 3 and must be prefixed by RECORD.

target\_row Identifies the row number to be hidden

*Rules* When a row at a higher scroll level is hidden, any associated child rows are hidden as well. When the HideRow function is used, the target row is hidden but there is no impact to the underlying database tables.

Returns A Boolean indicating the success (True) or failure (False) of the call

### Example

```
If MY_LOCATIONS.EFFDT < &TARGET_DATE Then
    HideRow(RECORD.MY_LOCATIONS, CurrentRowNumber(),
    RECORD.MY_LOCATION_EMP, &I);
    End-If;
```

# HideScroll

*Description* This function is similar to HideRow except that rather than hiding a row, the complete scroll area is hidden including all data in the scroll and the scroll bar.

# Syntax

HideScroll (Scrollpath)

#### ScrollPath is defined as:

```
[RECORD.level1_recname, level1_row, ]
[RECORD.level2_recname,] level2_row, ]
RECORD.target_recname
```

#### where

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname. The leve1\_recname requires the RECORD prefix.

level1\_row This parameter indicates data at scroll level one. The value is an integer and can be a variable or a constant. The parameter must be specified when hiding scroll areas at level 2 or 3.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row This parameter represents data at scroll level 2. The value is an integer and can be a variable or a constant. The parameter must be specified when hiding a scroll at level 3.

Target recordname The target scroll area to hide. The target record may be on scroll level 1, 2, or 3 and must be prefixed by RECORD.

*Rules* HideScroll is usually implemented in the RowInit and FieldChange events.

Returns A Boolean indicating the success (True) or failure (False) of the call.

```
Example
If %Mode = "U" Then
If ActiveRowCount(RECORD.MY_LOCATIONS) = 0 Then
HideScroll(RECORD.MY_LOCATIONS);
End-If;
End-If;
```

# RowScrollSelect

Description RowScrollSelect uses the select record parameter to read data and place it into a scroll specified for a particular parent row. This function is similar to ScrollSelect. The difference between ScrollSelect and RowScrollSelect is that ScrollSelect uses the key hierarchy of the parent keys and automatically places child rows under their corresponding parent data within the scroll buffer. RowScrollSelect does not do this and requires that the SQL string be used to limit the rows loaded into the scroll to those of the parent row keys.

#### Syntax

```
RowScrollSelect (levelnum, scrollpath,
RECORD.sel_recname
[, sqlstr [, bindvars]]
[, turbo])
```

### ScrollPath is defined as:

```
[RECORD.level1_recname, level1_row,
[RECORD.level2_recname, level2_row]]
RECORD.target_recname
```

### where

levelnum Represents the level number of the target scroll area. This value can be 1, 2, or 3.

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname. The leve1\_recname requires the RECORD prefix.

level1\_row Specifies the scroll level 1 row. The value is an integer and can be a variable or a constant. The parameter must be specified when the target record name is on scroll level 2.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row This parameter represents data at scroll level 2. The value is an integer and can be a variable or a constant. The parameter must be specified when the target record name is on scroll level 3.

target recordname Target record name appears at the lowest scroll level. The target record name must be prefixed by RECORD. Target record may be on scroll level 1, 2, or 3. When the target record is on scroll level 2, the target record name must be prefixed with the RECORD.level1\_recname, level1\_row parameter. When the target is on scroll level 3, the target record name must be prefixed with the RECORD.level1\_recname, level1\_row and the RECORD.level2\_recname, level2\_row.

RECORD.sel\_recordname Specifies the record or view from which data can be retrieved. Sel\_recordname can be the same as target\_recordname. One characteristic of this parameter is that it enables target rows to be loaded into a record with only those fields required in the scroll area in addition to key fields.

sqlstr [, bindvars] The SQL string parameter requires the SQL WHERE and an optional ORDER BY clause. The WHERE clause is used to limit any child keys read to those of the parent row key. The ORDER BY clause can be used to sort the rows before data are loaded into the target scroll area. The SQL string can accept bind variables that can be used as part of the WHERE or ORDER BY clause. Bind variables can be regular or inline bind variables. The SQL string can include Meta-SQL functions.

turbo Improves performance of the RowScrollSelect function. The parameter is passed as a Boolean True when Turbo RowScrollSelect is used.

*Rules* RowScrollSelect does not arrange child rows under their related parent row keys. It is up to the WHERE clause in the SQL string to limit child rows to the parent record key. Select record should be defined and created using Application Designer and cannot be a Derived/Work record.

Returns Does not return a value

### Example

```
/* Loads the Direct Deposit Distribution record for the current Emplid */
For &I = 1 To ActiveRowCount(RECORD.DIRECT_DEPOSIT);
RowScrollSelect(2, RECORD.DIRECT_DEPOSIT, &I,
RECORD.DIR_DEP_DISTRIB, "WHERE EMPLID = :1
ORDER BY EFFDT", PERSONAL_DATA.EMPLID, True);
End-For;
```

# RowScrollSelectNew

Description RowScrollSelectNew resembles RowScrollSelect, except that RowScrollSelectNew marks records as New when they are loaded into the scroll area. RowScrollSelectNew does not automatically place child rows under their corresponding parent key within the scroll buffer. It requires that the SQL string be used to limit the rows loaded into the scroll to those of the parent key.

#### Syntax

```
RowScrollSelectNew (levelnum, scrollpath,
RECORD.sel_recname
[, sqlstr [, bindvars]]
[, turbo])
```

### ScrollPath is defined as:

[RECORD.level1\_recname, level1\_row, [RECORD.level2\_recname, level2\_row]] RECORD.target\_recname

#### where

levelnum Represents the level number of the target scroll area. This value can be 1, 2, or 3.

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname. The leve1\_recname requires the RECORD prefix.

level1\_row Indicates the scroll level 1 row. The value is an integer and can be a variable or a constant. The parameter must be specified when the target record name is on scroll level 2.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row Represents data at scroll level 2. The value is an integer and can be a variable or a constant. The parameter must be specified when the target record name is on scroll level 3.

target recordname The target record name appears at the lowest scroll level. The target record name must be prefixed by RECORD. The target record may be on scroll level 1, 2, or 3. When the target record is on scroll level 2, the target record name must be prefixed with the RECORD.level1\_recname, level1\_ row parameter. When the target is on scroll level 3, the target record name must be prefixed with the RECORD.level1\_recname, level1\_ row, and the RECORD.level2\_recname, level2\_row.

RECORD.sel\_recordname Specifies the record or view from which data can be retrieved. sel\_recordname can be the same as target\_recordname. One characteristic of this parameter is that it enables target rows to be loaded into a record with only those fields required in the scroll area in addition to key fields.

sqlstr [, bindvars] The SQL string parameter requires the SQL WHERE and an optional ORDER BY clause. The WHERE clause is used to limit any child keys read to those of the parent row key. The ORDER BY clause can be used to sort the rows before data are loaded into the target scroll area. The SQL string can accept bind variables that can be used as part of the WHERE or ORDER BY clause. Bind variables can be regular or inline bind variables. The SQL string can include Meta-SQL functions.

turbo Improves performance of the RowScrollSelectNew function. The parameter is passed as a Boolean True.

*Rules* RowScrollSelectNew does not arrange child rows under their related parent row keys. It is up to the WHERE clause in the SQL string to limit child rows to the parent record key. Select record should be defined and created using Application Designer and cannot be a Derived/Work record.

Returns Does not return a value

#### Example

```
RowScrollSelectNew(2, RECORD.MY_LOCATIONS,
RECORD.MY_LOCATION_EMP, RECORD.MY_LOC_EMPL_VW,
"WHERE SETID = :1 AND OPRCLASS = :2 AND LOCATION = :3",
MY_LOCATIONS.SETID, MY_LOCATIONS.OPRCLASS,
MY_LOCATIONS.LOCATION, True);
```

# RowFlush

Description Used at the row level to remove a particular row of data from a scroll

Syntax RowFlush(scrollpath, target\_row)

### ScrollPath is defined as:

```
[RECORD.level1_recname, level1_row,
[RECORD.level2_recname, level2_row]]
RECORD.target_recname
```

where

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname. The leve1\_recname requires the RECORD prefix.

level1\_row Indicates the scroll level 1 row. The value is an integer and can be a variable or a constant. The parameter must be specified when the target record name is on scroll level 2.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row This parameter represents data at scroll level 2. The value is an integer and can be a variable or a constant. The parameter must be specified when the target record name is on scroll level 3.

target recordname The target record name appears at the lowest scroll level. The target record name must be prefixed by RECORD. The target record may be on scroll level 1, 2, or 3. When the target record is on scroll level 2, the target record name must be prefixed with the RECORD.level1\_recname, level1\_ row parameter. When the target is on scroll level 3, the target record name must be prefixed with the RECORD.level1\_row, and the RECORD.level2\_recname, level2\_row.

target\_row Identifies the row number to be removed from the specified target scroll area

*Rules* RowFlush does not remove rows from the database; it only removes them from the panel scroll buffer. To remove records from the panel scroll buffer as well as from the database, the DeleteRow function can be used because it performs both operations.

Returns Does not return a value

#### Example

```
If EMPLID <> PERSONAL_DATA.EMPLID Or
    EFFDT <> DIRECT_DEPOSIT.EFFDT Then
    RowFlush(RECORD.DIRECT_DEPOSIT, CurrentRowNumber(),
    RECORD.DIR_DEP_DISTRIB, CurrentRowNumber());
End-If;
```

# UpdateValue

*Description* UpdateValue is commonly used in a scroll area to update the value of a field using a value parameter.

#### Syntax

```
UpdateValue (Scrollpath, target_row,
[recordname.] fieldname, value )
```

### ScrollPath is defined as:

[RECORD.level1\_recname, level1\_row, [RECORD.level2\_recname, level2\_row]] RECORD.target\_recname

### where

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname. The leve1\_recname requires the RECORD prefix.

level1\_row Indicates the scroll level 1 row. The value is an integer and can be a variable or a constant. The parameter must be specified when the target record name is on scroll level 2.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row This parameter represents data at scroll level 2. The value is an integer and can be a variable or a constant. The parameter must be specified when the target record name is on scroll level 3.

target recordname The target record name appears at the lowest scroll level. The target record name must be prefixed by RECORD. The target record may be on scroll level 1, 2, or 3. When the target record is on scroll level 2, the target record name must be prefixed with the RECORD.level1\_recname, level1\_ row parameter. When the target is on scroll level 3, the target record name must be prefixed with the RECORD.level1\_recname, level1\_ row, and the RECORD.level2\_recname, level2\_row.

target\_row Identifies the row number in the specified target scroll area be to updated

[recordname.] fieldname The name of the field to be updated on the target row. Recordname is used when the function call is made from a record definition that is not the same as the recordname. Fieldname can reside on scroll level 1, 2, or 3.

Value Identifies the variable, constant, or record field that is moved to the corresponding target record

*Rules* The data type of the value parameter must be of a type compatible with the record field. The UpdateValue function updates the value of the field in the scroll. If the panel is canceled, no changes are written to the database.

#### *Returns* Does not return a value

```
Example
&NEW_DATE = %Date;
If EFF_STATUS <> "A" Then
For &I = ActiveRowCount(RECORD.DIRECT_DEPOSIT)
To 1 Step - 1;
UpdateValue(RECORD.DIRECT_DEPOSIT, CurrentRowNumber(),
RECORD.DIR_DEP_DISTRIB, &I,
DIR_DEP_DISTRIB.LAST_UPDATE_DATE, &NEW_DATE);
End-For;
End-If;
```

#### TotalRowCount

*Description* Produces the aggregate number of rows in a scroll area including deleted rows

Syntax TotalRowCount (Scrollpath)

#### ScrollPath is defined as:

[RECORD.level1\_recname, level1\_row, [RECORD.level2\_recname, level2\_row]] RECORD.target\_recname

where

RECORD.level1\_recname Represents the path to the target scroll area. When the target record is on scroll level 2, this parameter must precede the target\_recordname. The leve1\_recname requires the RECORD prefix.

level1\_row Indicates the scroll level 1 row. The value is an integer and can be a variable or a constant. The parameter must be specified when the target record name is on scroll level 2.

RECORD.level2\_recname Represents the path to the target scroll area. When the target record is on scroll level 3, the target\_recordname must be preceded by a RECORD.level1\_recname and RECORD.level2\_recname.

level2\_row This parameter represents data at scroll level 2. The value is an integer and can be a variable or a constant. The parameter must be specified when the target record name is on scroll level 3.

target recordname The target record name appears at the lowest scroll level. The target record name must be prefixed by RECORD. The target record may be on scroll level 1, 2, or 3. When the target record is on scroll level 2, the target record name must be prefixed with the RECORD.level1\_recname, level1\_ row parameter. When the target is on scroll level 3, the target record name must be prefixed with the RECORD.level1\_row, and the RECORD.level2\_recname, level2\_row.

*Rules* TotalRowCount is similar to ActiveRowCount except that TotalRowCount includes deleted rows. Rows that are marked as deleted remain in the buffer until all system updates have been performed.

*Returns* A number that includes active as well as deleted rows.

#### Example

```
/* To obtain total number of rows at level 1 */
&TOTAL_ROWS = TotalRowCount(RECORD.MY_LOCATIONS);
/* Total number of rows at level 2 */
&TOTAL_ROWS_LEVEL2 = TotalRowCount(RECORD.MY_LOCATIONS,
CurrentRowNumber(), RECORD.MY_LOCATION_EMP);
```

# SQL functions

# SQLExec

*Description* Executes an SQL command passed as a string from a PeopleCode program. The SQL string can contain bind variables, subselects, and joins. Data elements appearing in a Select statement are returned to the PeopleCode program as output and can be stored in variables or record fields.

#### Syntax

SQLExec (sqlcmd, bindvars, output)

sqlcmd Represents an SQL string passed by the PeopleCode program. It can contain references to both regular and inline bind variables.

bindvars Bind variables are the data elements referenced in the SQL string. There are two types of bind variables, regular and inline. When regular bind variables are used, each requires a corresponding variable name that replaces the :n reference in the SQL string. These variables appear outside the double quotes as

```
variable-1 [, variable-2, variable-3 ...]
```

When inline bind variables are used, the variables are enclosed within the SQL string as

[:recordname1.]fieldname1 [, [recordname2.]fieldname2] ...

output Represents the column name (s) populated as a result of a Select statement. The output can be placed into variables or record fields. Each column selected requires a corresponding output variable or record field separated by commas. The two forms include

variable-1 [, variable-2, variable-3] ...

or

[:recordname1.]fieldname1 [, [recordname2.]fieldname2] ...

*Rules* SQLEXEC is one function where unpredictable results can occur if rules are not followed. Because SQLEXEC bypasses the Application Processor and heads directly to the database, no evaluation of the SQL string contained within quotes is performed. Record fields used as inline bind variables or output variables are evaluated by the Application Processor when they are not contained in the SQL string. When PeopleCode containing SQLEXEC statements are entered into the PeopleCode editor, any undefined record fields are represented by an error message during the syntax check or save operation. SQLExec statements containing inline bind variables are the exception. Because an inline bind variable is enclosed in quotes, an SQL statement which contains incorrect inline bind variables generates a runtime error message. A previously undefined output variable is created at runtime and does not generate an error. A SQLExec Select statement retrieves one row of data only. When multiple rows are selected, only the first row is actually returned.

The maximum number of output variables when using Select is 64.

With SQLExec, Updates, Inserts and Deletes can be performed but can only be done in the following events:

SavePreChg WorkFlow SavePostChg

Application records referenced in a SQLExec statement require the PS\_ prefix.

*Returns* Returns an optional Boolean. A True indicates the function ran successfully.

#### Example

```
/* Using UPDATE with a regular bind variable */
SQLExec("Update PS_MY_LOCATIONS SET EFFDT = %1",
MY_LOCATIONS.EFFDT);
```

```
/* Using an inline bind variable */
SQLExec("Update PS_MY_LOCATIONS SET EFFDT =
:MY_LOCATIONS.EFFDT");
```

# **String functions**

#### Lower

*Description* Converts the uppercase characters of the field or variable to lower case and returns them as a String data type

Syntax Lower (string)

Rules Numeric, punctuation and other non-letter values are not changed

Returns A lowercase string

Example
&MY\_STRING = "THIS STRING BECOMES LOWER CASE";
&NEW\_STRING = Lower(&MY\_STRING);

# LTrim

*Description* Function is used to remove any leading characters identified in string2 from string1

Syntax Ltrim (string1 [, string2]) *Rules* When string2 is not supplied, all leading blanks from string1 are removed. When string2 is supplied, the function is terminated when characters found in string1 do not match those found in string2.

*Returns* A string with leftmost characters in string2 removed or blanks when string2 is not supplied.

#### Example

&STREET\_ADDRESS = ",##@&100 Main Street"; &STREET\_ADDRESS = LTrim(&MY\_STRING, ",.#@&"); /\* &STREET\_ADDRESS Now contains 100 Main Street \*/

#### RTrim

*Description* Function is used to remove any rightmost characters identified by trim\_str from the source string

Syntax RTrim (source\_str, [, trim\_str] )

*Rules* Works from right to left removing trailing characters defined in trim\_str. When trim\_str is not supplied, any rightmost blanks are removed.

*Returns* A string with leftmost characters in string2 removed or blanks when string2 is not supplied.

# Example

```
&DEPARTMENT_DESCR = "Software development & Web
Services,,,,,,";
&DEPARTMENT_DESCR = RTrim(&DEPARTMENT_DESCR, ",");
```

# Upper

Description Converts the characters appearing in a text string to upper case values

Syntax Upper (string)

*Rules* Characters such as numeric, punctuation, and other non-letter values are not changed

*Returns* Returns a string containing uppercase values.

```
Example
&LAST_NAME = "picard";
&LAST_NAME_SRCH = Upper(&LAST_NAME);
/* Value of &LAST_NAME_SRCH = PICARD */
```

# **Trace Control functions**

# SetTracePC

Description Controls PeopleCode Trace based on parameter values passed.

Syntax SetTracePC (n)

*Rules* Takes one parameter, which represents the trace settings used in producing the output trace file. When multiple trace options are required, each option number is added, and the sum is passed to the function. The options available to SetTracePC are shown in table E.2.

By default SetTracePC produces a file named DBG1.TMP in the Windows Temp directory. A unique file name can be specified if necessary, and this can be done from within the configuration manager trace option.

*Returns* Does not return a value

#### Table E.2 SetTracePC options

Option #	Description
1	This option traces the program that is executed. It includes options 64, 128 and 256 specified below.
2	Lists the entire program.
4	Displays the outcomes of assignments made to variables.
8	Identifies the values retrieved for all variables.
16	Identifies the contents used in the internal stack.
64	This trace option identifies when each program is started.
128	Identifies the calls made to external PeopleCode routines.
256	Identifies the calls made to internal PeopleCode routines.
512	Displays the value of parameters passed to a function.
1024	This option displays the values of parameters at the conclusion of a function cal

# **Transfer functions**

# SetNextPanel

*Description* SetNextPanel identifies a panel name that will be transferred control to when the operator activates the F6 or presses the NextPanel toolbar icon.

Syntax SetNextPanel (panelname)

*Rules* Verifies that the panel identified by panelname is available on the current active menu

*Returns* Returns an optional Boolean value based on the success or failure of the function call

# Example

```
If &RETURN_CODE = 0 Then
   SetNextPanel("MY_APPLCTN_TBL");
Else
   SetNextPanel("MY_USER_TBL");
End-If;
```

# TransferPanel

*Description* Transfers control to the next panel in the panel group, the panel name supplied to the function, or to the panel identified by a previous SetNextPanel function.

Syntax TransferPanel ( [panel\_name] )

Or TransferPanel ( )

*Rules* The panel transferred to must exist in the current panel group. When the function is called from events outside of save processing (SavePreChg, SavePostChg), any PeopleCode statements following the TransferPanel function are not executed and processing is halted.

*Returns* Returns an optional Boolean value based on the success or failure of the function call.

# Example If &RETURN\_CODE = 0 Then SetNextPanel("MY\_APPLCTN\_TBL"); Else SetNextPanel("MY\_USER\_TBL"); End-If; TransferPanel(); /\* Can also be written \*/ If &RETURN\_CODE = 0 Then &NEXTPANEL = "MY\_APPLCTN\_TBL"; Else &NEXTPANEL = "MY\_USER\_TBL"; End-If; TransferPanel(&NEXTPANEL);

# **META-SQL FUNCTIONS**

Meta-SQL functions are used in SQL strings. They expand in these strings to become platform-specific parameters in the SQL statements. SQL strings are used in the SQLExec as well as scroll functions that accept an SQL string. Meta-SQL can also be implemented when constructing dynamic views or Application Engine statements.

Function	Description
%CurrentDateIn	This is an In function that becomes a platform-specific SQL string. The string can be used to represent current date in a Select, Update, or Insert statement.
%CurrentDateOut	An $\mathtt{out}$ function that can be used as the current date in the $\mathtt{select}$ clause of an SQL string
%CurrentDateTimeIn	An In function that becomes a platform-specific SQL string. The string can be used as a DateTime value in a Select, Update, or Insert statement.
%CurrentDateTimeOut	%CurrentDateTimeOut is an out function that can be used as the current DateTime value in the select clause of an SQL string.
%CurrentTimeIn	This is an In function that becomes a platform-specific SQL string. The string is used as current time in a select, Update, or Insert statement.
%CurrentTimeOut	An out function that can be used as the current time in the Select clause of an SQL string
%DateAdd	Returns a date after adding the add_days parameter to date_from.
	syntax: %DateAdd (date_from, add_days)
	add_days is an integer that can have a negative value and is added to date_from.
%DateDiff	Identifies the difference between two dates
	syntax: %DateDiff (date_from, date_to)
	The difference between date_from and date_to is returned as an integer value. When a date literal is used, it must be passed as a %DateIn. Example
	&Difference = %DateDiff(INCIDENT_DT, CLOSE_DT);
	<pre>&amp;Difference = %DateDiff (CLOSE_DT, %DateIn('1999-05-31'));</pre>
%DateIn	An In function that becomes a platform-specific SQL string. The function accepts a date value parameter or a date literal in the format YYYY-MM-DD. %DateIn is used in SQL statements such as Select, Insert, and Update that require a date bind variable or date literal. syntax %DateIn(date)
%DateOut	An out function that can be used as the date in the select clause of an
Dateout	SQL string syntax: %DateOut(date)
°	
%TimeIn	This is an In function that becomes a platform-specific SQL string. The string is used as the time value in a select, Update, or Insert statement. The time parameter passed can be a time variable or a literal in the form hh:mm:ss.sssss [{AM PM}].

Table E.3 Selected Meta-SQL functions

Function	Description
%TimeOut	An $\mathtt{Out}$ function that can be used as the time in the $\mathtt{select}$ clause of an SQL string
%Substring	This function references only the portion of the string identified by source_ str. The starting position is identified by start and is relative to 1. Length rep- resents the number of characters to be referenced. <code>%substring</code> can be used to extract or compare a selected area of a string. syntax %Substring (source_str, start, length)
%TrimSubstr	This function is similar to <i>Ssubstring</i> and can be used to extract or com- pare a selected area of a string. The difference is that any trailing blanks in the string referenced by source are removed from the target substring. syntax %TrimSubstr (source_str, start, length)

Table E.3 Selected Meta-SQL functions (continued)

#### APPENDIX F

Application Engine functions

Eight basic functions or macros can be utilized in Application Engine statements: &BIND, &CLAUSE, &CLEARCURSOR, &EXECUTE, &MSG, &&RECORD, &ROUND, and &SELECT. We've already covered the most common functions in our exercises (&BIND, &MSG, and &SELECT). The only macro is &&RECORD.

#### &BIND

*Purpose* Retrieves an individual field value from the cache record.

Syntax &BIND(cache\_field [, NOQUOTES] [, NOWRAP] [, STATIC])

*Rules* The &BIND function can be used almost anywhere in an SQL statement. It cannot be used in a Select statement Result Set field list.

A character field is returned enclosed in quotes unless the optional NOQUOTES parameter is used.

Date fields are automatically enclosed (or "wrapped") within the %DATEIN or %DATEOUT Meta-SQL functions unless the optional NOWRAP parameter is specified.

When the STATIC parameter is specified, Application Engine resolves the &BIND variable before compiling the SQL statement. This is useful when creating dynamic SQL statements.

```
Example
&SELECT (COUNTER)
SELECT COUNT (*)
FROM PS_&BIND (RECNAME, NOQUOTES, STATIC)
```

The example is the same used in exercise #3 of our tutorial. When using a RECNAME of JOB, the following SQL statement is compiled: SELECT COUNT(\*) FROM PS\_JOB

The value of JOB is not enclosed in quotes due to the NOQUOTES parameter and can therefore be concatenated properly with the PS\_ prefix. The STATIC parameter tells Application Engine to resolve the &BIND variable before compiling the statement.

The &SELECT portion of this statement is described in the &SELECT section.

#### &CLAUSE

*Purpose* Similar to a COBOL copybook (or #Include in SQR). When used in a statement it is replaced with the contents of the Application Engine statement specified in the &CLAUSE function. One of the main uses of the &CLAUSE function is retrieving predefined column lists and substituting them in the calling statement. This is useful for Select lists or Insert statements. There are several parameters that can be used with &CLAUSE to increase its' flexibility.

#### Syntax

```
&CLAUSE(product, application, section, step, type [,parm1] ... [,parm9])
```

*Rules* &CLAUSE must point to a valid statement designated by the fully qualified Application Engine statement name of product, application, section, step, and type. No validation is performed. Any errors will be recognized at run-time.

The actual clause section that's retrieved may have optional &P(n) parameter variables embedded within it. A parameter value must be passed to it with the &CLAUSE function. Up to nine parameters may be passed. Several symbolic parameters may be used in an &CLAUSE function:

COMMA Since a physical comma (or ',') would not be interpreted as an actual parameter, the COMMA symbolic can be used.

&SPACE This symbolic represents a blank or space. If the retrieved clause section uses &P(n) but isn't required a space can be passed to resolve them using &SPACE.

&RPAREN A right parenthesis may be passed to the specified "clause section." Once again, the symbolic must be used. A physical right parenthesis would not be interpreted as a parameter that needs to be passed.

One of the primary uses of parameters in a &CLAUSE function is to pass a synonym to be used as a column prefix.. The table synonym must be passed with a period like 'A.' or 'B.' —without the quotes. The &CLAUSE function would then return the columns with the desired prefixes to your current statement.

*Example* Assuming we defined a "clause section" for product = AE, Appl ID = SAMPLE, section = COMMON, step = CITYINFO, and statement type = S. The column list is entered in the statement text box as follows:

```
&p(1)CITY
,&p(1)STATE
,&p(1)ZIP
```

The &CLAUSE function allows us to substitute the above column list anywhere in our program. Using parameters, we can tailor the column list to our particular needs with prefixes. You'll notice the &P(n) prefix variable contains a '1' for all three columns. This means the first parameter passed in the &CLAUSE function is used as the prefix for all three columns. Using &p(n), the n represents the n<sup>th</sup> parameter passed in the &CLAUSE.

```
INSERT INTO ps_user_cityinfo
( &CLAUSE(AE, SAMPLE, COMMON, CITYINFO, S, &SPACE) )
SELECT &CLAUSE(AE, SAMPLE, COMMON, CITYINFO, S, A.)
FROM ps_temp_cityinfo A
```

The &CLAUSE is used twice: Once for the Insert Column list, which doesn't allow prefixes, and once for the Select Column list, which in this case uses a prefix of 'A'. When the SQL statement is resolved, it appears as:

```
INSERT INTO ps_user_cityinfo
(CITY, STATE, ZIP)
SELECT A.CITY, A.STATE, A.ZIP
FROM ps_temp_cityinfo A
```

Also, see the &&RECORD macro for similar functionality.

# &CLEARCURSOR

*Purpose* A re-used statement may need to be recompiled during execution of the program. The &CLEARCURSOR function accomplishes this and resets any &BIND variables in the statement that use the STATIC option.

```
Syntax
&CLEARCURSOR([product, ] [application, ] section, step,
type)
```

*Rules* This function must be located at the start of the statement. There may be no other functions or commands in the statement.

```
Example &CLEARCURSOR(BI, BIIVC000, DUEDATE, SETDATE, D)
```

This recompiles the DO Select statement in the SETDATE step. The step is found in the DUEDATE section of the billing application BIIVC000.

**NOTE** Refer to the section describing the Statement Definition panel in appendix F for an explanation of re-used statements.

#### &EXECUTE

*Purpose* Database-specific commands may be executed with this function. Generally, this means any SQL statement that cannot be executed directly using the Update/Insert/Delete SQL statement type.

#### Syntax

```
&EXECUTE( [/] )
command_1 {; | /} ...
command_n {; | /}
```

*Rules* The Update/Insert/Delete statement type must be used. This function must be located at the start of the statement. No other functions or commands may exist in the statement.

Application Engine expects each command within the &EXECUTE function to be delimited with a semi-colon. The optional forward slash (/) parameter overrides this convention and allows the use of a procedural language such as Oracle's PL/SQL to be used. Since the commands within a PL/SQL block are normally terminated by a semi-colon, the forward slash override avoids any conflict. The forward slash would then be required at the end of the &EXECUTE statement.

#### Example

```
&EXECUTE(/)
declare
    ctr integer:= 0;
begin
    while ctr = 0 loop
        ctr = ctr + 1;
        update ps_installation_ar set st_id_num = ctr;
    end loop;
end;
/
```

The forward slash (/) tells Application Engine to execute the entire PL/SQL block. No conflicts result due to the semi-colon.

#### &MSG

*Purpose* The &MSG function writes a message to the message log.

```
Syntax
&MSG([Message_Set_Number], Message_Number, [Parm_1],....
[Parm_n])
```

*Rules* The &MSG function always uses an SQL statement type of Update and must be the first and only function or command in the statement.

Example &MSG(,1,'Hello World')

The example is the same as that used in exercise #1. Since the Message Set Number is excluded, it defaults to the Message Set Number specified on the Application Engine definition panel. Message Number 1 is passed a string value of "Hello World". This string value is used in place of the %1 substitution variable defined in the message catalog entry.

# &&RECORD

*Purpose* The &&RECORD macro inserts all the field names of the specified record into your statement. The optional parameter can be used to assign a column synonym when the entire record is required in your statement(s). This is a quick alternative to the &CLAUSE function.

Syntax &&RECORD(record [, parm\_1] )

Rules You must use a valid RECNAME.

# Example

INSERT INTO ps\_customer\_tao SELECT &&RECORD(CUSTOMER) FROM ps customer

Using &&RECORD, the Select statement uses all the columns in the Customer record as they exist in Application Designer. This example assumes the CUSTOMER\_TAO record matches the Customer record exactly.

#### &ROUND

*Purpose* When Multi-Currency is activated, this function can be used to round numeric fields to the currency precision specified under Define General Options.

Syntax &ROUND(field) Rules The Multi-Currency option must be specified. To set the Multi-Currency option Go →PeopleTools →Utilities →Use →PeopleTools Options To set the currency precision: Go →Define Business Rules →Define General Options →Use A-D → Currency Code

#### Example

```
UPDATE ps_user_tmp
SET USER_AMT1 = &ROUND(USER_AMT1)
```

This example updates the table with the USER\_AMT1 value rounded to the appropriate currency precision.

#### **&SELECT**

*Purpose* Updates the cache field with the value assigned by the corresponding SQL Select statement

#### Syntax

```
&SELECT(cache_field_1 [,cache_field_2] [,cache_field_x] )
SELECT field_1 [,field_2] [,field_x]
```

*Rules* &SELECT is used in tandem with an SQL Select statement immediately following.

- The number of cache fields must match the number of fields in the SQL Select.
- The datatypes of corresponding cache and SELECT fields must match.
- If NO rows are returned by the SQL Select statement, the cache fields are assigned a value of zero or blank, depending on the datatype.

#### Example

```
&SELECT (COUNTER)
SELECT COUNT(*)
FROM PS_PERSONAL_DATA
```

The example is the same as that used in exercise #2. The record count is selected from the PERSONAL\_DATA table (lines 2 and 3). The &SELECT function (line 1) assigns the record count to the cache field COUNTER.

Licensed to James M White <jwhite@maine.edu>

# index

#### System variables

SQR variables #prcs\_run\_status 639 \$prcs\_oprid 646 \$prcs\_process\_instance 637 \$prcs\_run\_cntl\_id 646 PeopleTools variables % PanelGroup 386 %BPName 276 %Date 276, 379 **%DATEIN 813** %DATEOUT 813 %DateTime 276 %DbName 276 %DbType 276 %EmployeeId 276 %Import 45, 276 %KeyEqual 421 %Language 276, 282 %Market 276 %Menu 276 %MessageAgent 276 %Mode 277, 300 %OperatorClass 277 %OperatorId 277, 321 %Panel 277 %PanelGroup 277 %SQLRows 277, 335 %Time 277 %WLInstanceID 277 %WLName 277

Application Engine variables &&RECORD 1060 &BIND 786, 813, 816, 821, 827, 834, 945, 1056 &CLAUSE 1057 &CLEARCURSOR 1058 &EXECUTE 1059 &MSG 802, 806, 823, 839, 859, 1060 &ROUND 1060 &SECTION 875, 886, 949 &SELECT 811, 816, 821, 865, 1061

#### Α

Abort Application 784 Absent status (Upgrade flag) 449 Access groups 67 Access Profile 54 ACTION (COBOL) 721 Action (Panel Group) 449 Action Add (Upgrade) 450 Action Delete (Upgrade) 450 Action Mode (Panel Group) Add 176 Correction 176 Data Entry 176 Update/Display 176 Update/Display All 176 Action Replace (Upgrade) 450 **ACTION-COMMIT 729** 

ACTION-FETCH 732 **ACTION-SELECT 730** Activate event 420 active scroll area 303 ActiveRowCount (PeopleCode function) 304, 353 Add 16, 176, 220, 221, 224 Add Search Record 153 AddKeyListItem (PeopleCode function) 387 AddToDate (PeopleCode function) 378 administration tools 32-68 AE APPL TBL 777 AE\_MESSAGE\_LOG 897, 906 AE\_REQUEST 889, 891, 893, 894, 895, 896, 897, 898, 902, 906 AE\_RUN\_CONTROL 782, 788, 929, 946 AE\_SECTION 873, 875, 880, 882, 893 AE SECTION TBL 777, 778 AE\_STEP\_TBL 777, 778 AE STMT B TBL 779 AE\_STMT\_TBL 777,779 AEADHOC 804, 815, 825 All (PeopleCode function) 380 Alter comments to script 214 Alter Tables 210 Alter/Create scripts 453

alternate search key 93, 145, 152,206 ANY (Data Type) 272 API Aware 624, 636, 652, 654 API Aware flag 624, 640 API Aware process 634, 640 API Aware program 639 API files 635 API programs 683 Application Designer 1, 19, 58, 69, 75, 249-256, 259, 260, 475, 739, 793, 896 PeopleCode 263 Application Engine 250, 769, 771,908 advantages 772 analyzer 948 analyzing programs 947 compared to SQR 805, 826, 848, 870, 886 debugger 966 definition panels 779 definition tables 777 disadvantages 772 macros 963 main components 776 messages 805 PeopleCode 963 program 802 restarting a process 946 application folder tab 780 Application Library 958 application menus 113 Application Processor 69, 216, 218, 220, 223, 225, 226, 229, 230, 234, 240, 244, 247, 257, 261, 272, 293, 333, 337, 553, 612 Application Reviewer 401, 408 Application Run Control record 574, 600, 643 application server 6, 9, 126 Application upgrade 438 architecture 4 arithmetic operators 288

array objects 422 arrays in a loop 423 multi-dimensional 422 populating 422 removing items 423 Audit 96 Audit Flags 451 authorizing users 118 Auto Update 96 Automatic Insert option 477 AutoSelect 342

#### В

Background Disconnect Interval 48 backward compatibility 416 bar items 14, 114 BEA Systems 5 Bind Data 724, 763 Bind Setup 724, 763 bind variables 333, 731, 733, 734, 735, 786 BOOLEAN (Data type) 272 Boolean constants 275 Boolean operators 289 Boolean value 335 Break (PeopleCode statement) 282, 284 Break at Cursor 406 Break at Start 402, 403 BRIO Technology 569 Build (records) 211, 215 build options 210 building database tables 204 built-in functions (PeopleCode) 376 **Business Component 250 Business Interlink 250** Business Process Map 48

# С

cache (Application Engine) Process Request panel 824 cache fields 807-816 assigning values 811 values 813 cache record 771, 780, 788, 789, 806, 807, 824, 893 custom 793 USER AET 817 calling functions 260 Cancel 244, 248, 307, 320, 385 catalog tables 198 C-callable library 396 Change Control 450 Changed \* (Upgrade) 449 Changed/Unchanged (Upgrade) 449 Char (PeopleCode function) 377 character field 296 Check Box 460 child key 341, 362 child row 341, 365 Citrix 4 class of operators 47, 48 client 9 COBOL 713, 753, 771 trace 761 trace file 758 trace file contents 763 using SQL 716 COBOL Analyzer 766 COBOL Call Structure listing 766 COBOL program 715, 784, 800 COBOL statement timings 764 Column Length Options 212 command button 161, 164 command line 666 command-line parameters 645 Comment (Application Engine) 777, 785 comment (PeopleCode) 271 COMMIT 729 commit 247, 309, 781 commit limit 451 Commit-Transaction function 640

communicating errors 639 communication area 722 Compare Type 446 comparison operator 288, 335 compilation errors 558 concatenation (PeopleCode) 287 Configuration Manager 1, 11, 528, 628, 648, 758 Application servers 13 Client Setup 14 Common 13 Crystal 13 Database 13 Display 12 nVision 13 Process Scheduler 13 Remote Call 14 Startup 12 Trace 13 Workflow 13 CONNECT 728 constants 275 contiguous setup list strings 727 control display 229 control statements 279 conversion functions 377 Copy dialog panel 451 Copy Options 84 Copy process 450 Copy to Clipboard push button 580 Correction (Action Mode) 16, 153, 176, 177, 178, 225 Create Indexes 210 Create Tables 210 Create Views 210 creating a custom panel 498-513 creating a custom record 495-498 Critical Database Update 782 cross reference files 765 cross references 30 Cross-Reference Utilities 28 Crystal Report (.rpt) 576, 708 Crystal Reports Process 708

Ctrl-F7 82 CurrEffdt (PeopleCode function) 378 CurrEffSeq (PeopleCode function) 379 CurrentRowNumber 352, 355 CURSOR 723 custom (Field Format) 86 CUSTOM (PTPSQLRT parameter) 84 customization 431, 473 Customization Upgrade 444

# D

data access 332 Data Administration 100, 204 data conversion 337 Data Definition Language (See DDL) Data Entry (Action Mode) 176 data entry (Application Processor) 236, 303 data model 591 Data Mover 1, 33, 717, 719, 742 scripts 719 data retrieval 225, 300 Data Trace 409 data types 271 ANY 272 BOOLEAN 272 **DATE 272** DATETIME 272 NUMBER 272 **OBJECT 272** STRING 272 **TIME 272** database catalog tables 198 keys 148, 205 object modeling 202 security 660 type of compare 449 views 204 DATE (Data type) 272 Date (PeopleCode function) 377

date comparison 136 Date/Time 377 defaults 576 parameters 576 DatePart (PeopleCode function) 272 DATETIME (Data Type) 272 DateValue (PeopleCode function) 377 DBG1.tmp 528 DDDAUDIT 201 DDL 100, 207, 208 DDL Model 203 Debug 781 debugging 400 breakpoint 401, 427 viewing data 407 debugging tools 399-415, 956, 966 Application Reviewer 401 WinMessage 400 decimal number setup 726 decision logic 849-870 Declare Function 395, 396 Default Panel Control 96 Default processing 233 Define List button 411 Define-Prcs-Vars (SQR) 635, 637 Delete (COBOL) 733, 912 DELETE (SQLExec) 336 Delete Row (F8) 16, 351, 367 DeleteRecord (PeopleCode function) 383 DeleteRow (PeopleCode function) 347, 355, 370, 381 Department security 548, 664 Department table 38 Derived records 84, 155, 160 Derived/Work record 154, 504, 506, 512, 557 Descending Key 94, 206 Description 623 Detail Trees 66 development objects 250 development steps 889

INDEX

development tab 443, 564 DiscardRow (PeopleCode function) 301 DISCONNECT (COBOL) 728 DISCONNECT ALL (COBOL) 728 Display Control 105, 501 display-only 104 DispOnly 49 DLL 397 DMS script 719, 727, 734, 742 DO section 784, 834, 846, 927, 949,950 DO Select 777, 784, 785, 833, 841,854 DO Until 777, 785 DO When 777, 785, 864, 865, 918, 923 DO While 777, 786 DoCancel (PeopleCode function) 385 DoSave (PeopleCode function) 385 DoSaveNow (PeopleCode function) 521 dot notation 417 Drop Column Options 212 drop-down list 102, 131, 460 DUAL 864,870 duplicate keys 306 Duplicate Order Key 93, 206 dynamic prompt 137, 154 dynamic sections 871-886 Dynamic SQL 814 dynamic views 84 DYNSECTN section 925

# Ε

Edit Box 102, 460 Edit Run Control panel 585 edit search fields 223 Edit Type 97 editable fields 384 editing values 260 EFF STATUS 134, 135, 136 EFFDT 134, 135, 136, 177 effective date 134, 176, 178, 378 embedded SQL 332-339, 716 EMPLMT\_SRCH\_GBL 661 Employee ID 54 End-Function 394 End-If 280 ERP 429 error 298, 304, 307, 309, 320, 326, 579 duplicate key 306 ERROR (COBOL) 728 error handling 313-331 error-handling (COBOL) 716 **ERR-SECTION 729** Evaluate (PeopleCode statement) 281 event-driven languages 257 Execute options 211 Execution Trace 409 Explain (Messages) 315 Export 35 expressions 286 external function (PeopleCode) 393, 559 external Non-PeopleCode functions 396

# F

-f flag (SQR flag) 655
F7 82
F8 239
Fast Security 669
FETCH 731
Fetch 418, 730, 732
FetchValue (PeopleCode function) 357
FieldChange (PeopleCode event) 26, 261, 299, 303, 352
FieldChanged (PeopleCode function) 382
FieldDefault 26, 95, 261, 295, 296, 302, 404
FieldEdit 26, 261, 298, 303, 327 FieldFormula 26, 261, 295, 296, 393, 395, 556, 557 Field-Level security 660 Fieldname 274 fields 20, 78 character 296 definitions 78, 255 editable 384 hidden 384 invisible 384 label 462 modification 236 numeric 296 File Layout 252 File object 417 File Overwrite Options 214 file-handling methods 417 Find Object References 468, 536 Fit/Gap Analysis 72 Folder Tab Label 611 font settings 270 For (PeopleCode statement) 283 From Search field 94 From Search keys 149 FUNCLIB 556 function categories 178 (See also appendix E) built-in 376 Cancel 385 conversion 377 Date/Time 377 Effective Date/Sequence 378 external 393 internal 389 Logic 380 Math 380 Message Catalog/ Display 314 overview 375 Panel buffer 381 Panel control 384 Panel transfer 387 Process Scheduler 388 Save 385 scroll 365

function categories*(continued)* Sequence 378 String 386 Trace 413 function declaration 395 Function keys F7 82 F8 239 function libraries 278, 297, 374– 398 Function statement (PeopleCode) 389

# G

General Attributes 48 general environment 255 General tab 105, 111, 116 Get-As-Of-Date 648 Get-Run-Control-Parms 637 Get-Values procedure 646 Global Time Zone 711 global variables 277, 278 granting security access 523 Gray (PeopleCode function) 384 Grids 23 column widths 192 copy data 190 creating on a panel 193 freezing columns 192 from spreadsheets 191 into spreadsheets 190 row heights 192 sorting 190 Group boxes 22

#### Н

hidden fields 384 Hide (PeopleCode function) 384 HideRow (PeopleCode function) 359 HideScroll (PeopleCode function) 360 HTML 250, 253

INDEX

HTML definitions 253 HTML Document (.htm) 708

# I

Icons 76-78 If (PeopleCode statement) 280 If-Then-Else 279 Ignore Error (Application Engine) 784 **IGNORE DUPS 37** image definition 253 Images 23 Import 36 Import Header 39 Import Manager 37, 276 indexes 208 Initiated (Process Scheduler Status) 579, 637, 639 inline bind variables 334, 339 Input dialog box 95, 145 input parameters 643, 644, 697 inquiry panel 187, 188 INSERT 336, 337 Insert 734, 735 Insert Row (F7) 16, 351 InsertRow (PeopleCode function) 383 **INSTALLATION 668** Int (PeopleCode function) 381 integrating SQC files 648 internal functions 389 Internet Client 249 invisible fields 384 Issue Message 781 item name 546 ItemSelected (PeopleCode event) 263

# J

Java applets 7 job 688 Job Definition (Process Scheduler) Job Description 694 Job Priority 694 Job Run Mode 694 Job Definition (Process Scheduler) (continued) Parallel 694 Process Class 694 Recurrence Name 695 Run Always 695 Serial 694, 695 Server Name 694 Job Definitions panel group 695 job stream 688, 690, 691 Jolt 7 Jolt Internet Relay 7 Jolt Relay Adapter 7

# Κ

Keys 93 Alternate Search Key 93 Descending Key 94 Duplicate Order Key 93 From Search Field 94 List Box Item 94 Search Key 94 Through Search Field 94

# L

Label 103, 104 label type RFT Long 463 RFT Short 463 Text 512 language elements 268 Language Preference 54 LASTUPDOPRID 449 layout (Panel Definition) 105 leading characters, removing 386 Len (PeopleCode function) 387 level one (Scroll Bars) 170 level one record 342 level parameter 355 level two (Scroll Bars) 170 level zero (Scroll Bars) 171 level zero record 341 list box 145, 147, 152 List Box Item 94

local variables 277 Log Client Request 624 log file 35, 36, 410 log window 409 Logging Level 213 Logging Output 213 logic functions (PeopleCode) 380 LONG (Data Type) 335 Long Description 623 Long Edit Box 23, 460 loop 284, 354 Lotus 1-2-3 files (.wks) 708 Lower (PeopleCode function) 386 LTrim (PeopleCode function) 386

#### Μ

MAIN (Application Engine) 777, 798, 799, 808, 828, 839, 854 Mass Change 771, 784, 800 math functions (PeopleCode) 380 math operations (PeopleCode) 288 menu 14, 113, 117, 276, 544 actions 295 attaching panel group 899 modifying 515 pop-up 113, 242, 305 Problem Tracking 616 properties 116 security 666 standard 113 Menu Items 49, 108, 112, 113, 114, 119, 614 properties 115 Menu PeopleCode 263, 267 Menu Upgrade 565 Message catalog 316, 326, 330, 789, 790 Message channel 254 Message definition 253

Message log 771, 936 Message node 254 Message number 318, 330 Message set 316, 330, 790, 1060 Message severity 319 MessageBox 314, 316, 405 style parameters 314 Meta-SQL 272, 337, 772, 956, 962 Meta-Variables 963 Microsoft Excel (.xls) 708 Mod (PeopleCode function) 381 mode (Panel Group) 218, 220 modifying PeopleSoft COBOL 738-747 MsgGet (PeopleCode function) 330 MsgGetText (PeopleCode function) 330 multi-dimensional arrays 422 multiple rows 828–848 processing 829

# Ν

nested statements (PeopleCode) 280 Network security 660 Next in List 16 Next Panel in Group 16 NextEffDt (PeopleCode function) 379 No Auto Select 172, 173, 174 No Auto Update 174 Node Oriented Trees 66 None (PeopleCode function) 380 NOQUOTES 813, 821, 827, 1056 NOWRAP 813, 1056 n-tier 8 NUMBER (Data Type) 272 Number (Field Attribute) 80 numeric 275 numeric field 296

# 0

OBJECT (Data Type) 272 object group 60, 62 **Object Inspector** 77 object modeling 202 Object references 28, 536 Object security 58, 59 object types 58 **Object Workspace** 76 obsolete process definition 642 online help 13 Online panels 572 online security 660 Online Trace File 758 operation code 410 operation operands 410 operations hierarchy 288 operator 118, 126 operator class 47, 67, 118, 124, 126, 277 Operator ID 47, 53, 55, 56, 635 Operator Password 54 **Operator Preferences** 63 operator security 900 OPRCLASS 118, 128, 149 OPRID 118, 127, 149 Output Destination 587, 707, 708 output file 655 Output Format 707, 708 Output Options 702 output options 707 Output Type 707 Output Variable 335 Output Window 76 Override Options 705

# Ρ

Panel buffer functions 381 Panel control functions 384 Panel definition 101, 256 panel design 165–197 Panel Design Icons 77 Panel Field properties 103

INDEX

Panel Group 24, 109, 112, 218, 247, 277, 419, 541, 546, 610, 624 adding a panel 485 attaching to menu 899 custom 513 definitions 256 display 232, 302 structure 419 upgrade 565 Panel Group Icons 78 panel layout 105, 170 Panel Properties 105 Panel Transfers tab 627 panels 22, 109 parameter descriptions 721 parameter list 319, 397 parameters 344, 603, 698 level (Scrolls) 355 MessageBox 316 style (Message) 316 Turbo (Scrolls) 347 parent data 365 parent key 342 parent record 91 parent row 341 partial key 145 Password 125 PeopleCode 26, 29, 44, 69, 100, 257, 520, 521, 891, 902, 903, 956 comments 271 data elements 273 debugging tools 399-415 editor 259, 269 embedded SQL 332-339 events 261 expressions 286 font settings 270 function libraries 374–398 language elements 268–292 PeopleSoft 8 416-427 RowSelect 315 runtime errors 261 scrolls 340-373 search 411

PeopleCode (continued) SQLExec commands 520 statements 278 toolbar icon 265 tools tables 289 Trace 413 PeopleCode Debugger 423 PeopleCode editor 269, 274, 334 PeopleCode event FieldChange 521 FieldDefault 26, 95, 261, 295, 296, 302, 404 FieldEdit 26, 261, 298, 303, 327 FieldFormula 26, 261, 295, 296, 393, 395, 556, 557 ItemSelected 263 PrePopup 27, 262, 305 RowDelete 239, 262, 303, 304, 327, 354, 355 RowInit 261, 296, 297, 302, 304, 354, 556 RowInsert 239, 262, 303, 354, 383 RowSelect 262, 301, 324 SaveEdit 262, 299, 307, 327, 380 SavePostChg 521 SavePreChg 521 SearchInit 262, 296, 297, 300 SearchSave 262, 300, 301 WorkFlow 521 PeopleCode functions 557, 560 (See also appendix E) ActiveRowCount 304, 353 AddKeyListItem 387 AddToDate 378 All 380 Char 377 CurrEffdt 378 CurrEffSeq 379 Date 377 DatePart 272 DateValue 377 DeleteRecord 383

PeopleCode functions (continued) DeleteRow 347, 355, 370, 381 DiscardRow 301 DoCancel 385 DoSave 385 FieldChanged 382 Gray 384 Hide 384 HideRow 359 HideScroll 360 InsertRow 383 Int 381 Len 387 Lower 386 LTrim 386 Mod 381 MsgGet 330 MsgGetText 330 NextEffDt 379 None 380 PriorEffDt 379 RecordChanged 382 RecordDeleted 383 RecordNew 383 Round 380 RowFlush 367 RowScrollSelect 363 RowScrollSelectNew 365 RowSelect 26, 232, 262, 301, 315, 324 RTrim 376, 386 ScheduleProcess 388, 394 SearchInit 26, 262, 296, 297, 300 SearchSave 26, 262, 300, 301 SetCursorPos 307, 327 SetNextPanel 387 SetTracePC 414 StopFetching 301 SUBSTR 338 Time 377 TimePart 272 TotalRowCount 372 TransferPanel 388 UnGray 385

INDEX

PeopleCode functions (continued) UnHide 384 UpdateValue 370 Upper 386 Warning 298, 304, 307, 309, 320, 327 WinMessage 324, 400 PeopleCode statements 278 Break 282 Declare Function 395 Else 280 End-Evaluate 283 End-If 280 Evaluate 281 For 283 Repeat 284 Return 396 True 279 When 281 When-Other 282 While 285, 286 PeopleSoft API 572 architecture 4 Configuration Manager 628 security 659 PeopleSoft 8 8, 249-256, 416-427, 702–711, 954–968 PeopleSoft Online security 660 PeopleSoft Query tool 662 PeopleTools 8 417, 710 Acrobat (.pdf) 708 array objects 422 Comma Delimited (.csv) 708 file object 417 HP format (.lis) 708 HTML documents format (.htm) 708 Line Printer Format (.lis) 708 Other (.lis) 708 output formats 708 panel group 419 Panel group PeopleCode 419 PeopleCode Debugger 423 Postscript (.lis) 708

PeopleTools 8 (continued) Process Request Monitor 703 **Process Scheduler** Manager 703 Process Scheduler Request Dialog 703 Process Scheduler Server Agent 703 Rowset 420 SQL object 418 SQR Portable Format (.spf) 708 PeopleTools catalog 199 (See also appendix C) **PSDBFIELDLANG 21 PSINDEXDEFN 22 PSKEYDEFN 22 PSMENUDEFN 25 PSMENUDEFNLANG 26 PSMENUITEM 26 PSMENUITEMLANG 26 PSPCMNAME 27** PSPCMPROG 27 **PSPNLDEFN 24** PSPNLFIELD 24 **PSPNLGDEFNLANG 25 PSPNLGROUP 25** PSPNLGRPDEFN 24 **PSPROGNAME 27 PSPROJECTDEFN 27 PSPROJECTITEM 27** PSPROJECTMSG 27 PSRECDDLPARM 22 **PSRECDEFN 21 PSRECDEFNLANG 22** PSRECFIELD 21 XLATTABLE 140 PeopleTools run control record 576, 583 PeopleTools upgrade 438 PERSONAL DATA 807 PERSONAL\_SRCH\_QRY 661 PGM\_NAME 718,723 Physical Application Server 6 platform dependent 338

platform independence 337, 338 platform specific 337 populate search fields 222 populating arrays 422 pop-up menu 25, 113, 117, 242, 305 PostBuild 420 PRCSAPI.sqc 635, 637 PRCSDEF.sqc 635 PRCSRUNCNTL 603, 606 PRCSRUNCNTL\_LC\_SBP 60 6 PreBuild (PeopleCode event) 420 PrePopup (PeopleCode event) 27, 262, 305 Previous in List 16 Previous Panel in Group 16 primary records 344 printer setup SQCs 711 PriorEffDt (PeopleCode function) 379 Problem Tracking (application) 34, 69, 309, 384, 393, 590 Problem Tracking menu 616 **PROCEDURE DIVISION 739** Process Class 623 Process Definition 113, 123, 617, 621, 622, 623, 629, 703, 903 API Aware 624 Description 623 Log Client Request 624 Long Description 623 Panel Group 624 Priority 623 Process Class 623 Process Security Groups 625 Recurrence Name 623 Run Location 623 Server Name 623 SQR Runtime 624 Process Definition dialog 621 Process Definition Options panel 626

1070

INDEX

Process Definition panel group 704 Process Detail panel 580 Process groups 53, 118, 121 Process Instance 635 Process Monitor 572, 578, 581, 632, 654, 749, 750, 754 status 579 process profiles 123 Process Request 684, 803, 821, 843, 934 assign cach values 824 Process Request detail 579 Process Request table 636 Process Run status 639 Process Scheduler 69, 113, 123, 388, 572, 574, 635, 642, 752, 803,903 Output Destination 576 PeopleCode support 710 Request dialog 576 Run Date/Time 576 Run Recurrence 576 Process Scheduler API 749 Process Scheduler functions 388 Process Scheduler Request 576, 587, 631, 702, 706 Process Scheduler security 702, 709 Process Scheduler terminology 703 Process Security groups 625, 666 process type 621 PROCESS INSTANCE 793, 794, 795, 929, 944 Processing (Process Status) 579, 637,639 **PROCESS-INSTANCE 750** ProcessRequest class 710 PROGCOUNT 21 Project Workspace 76, 958 projects 82 prompt 134, 136, 137, 138, 139, 240 prompt records 131 Prompt Table Edit 97, 238

Prompt Table No Edit 97 properties 93, 105, 111, 193, 196,611 prototype 74 PS\_EMPLMT\_SRCH\_GBL 66 4 PS\_EMPLMT\_SRCH\_US 533 PS\_FAST\_PERSGL\_VW2 671 PS\_PERS\_SRCH\_QRY 664 PS PRCSDEFN 642 PS\_PRCSDEFNGRP 642 PS\_PRCSDEFNPNL 642 PS PRCSDEFNXFER 642 PS SQLSTMT TBL 717 PSAUDIT 92, 451 **PSAUTHITEM 56** PSDBFIELD 200, 238, 448 PSIDXDDLPARM 199, 200 PSINDEXDEFN 22, 199, 200 PSKEYDEFN 22, 199, 200 PSLOCK 864, 870, 955 PSMENUDEFN 25, 199 PSMENUDEFNLANG 26 PSMENUITEM 26, 199 **PSMENUITEMLANG 26 PSOPERDEFN 321** PSOPRDEFN 277, 334 PSPCMNAME 27, 199, 289 PSPCMPROG 27, 199, 269, 290 PSPNLDEFN 24, 199 PSPNLFIELD 24, 199, 229, 642 PSPNLGROUP 25, 199, 228 PSPNLGRPDEFN 24, 199, 218 PSPRCSRQST 388, 636, 639, 642,750 PSPRCSRQST table 639 **PSPRCSRUNCNTL 576** PSPROGNAME 27, 199, 290 PSPROJECTDEFN 27 **PSPROJECTITEM 27** PSPROJECTMSG 27 PSPRSCRQST 638 PSRECDDLPARM 22, 199, 200, 208

PSRECDEFN 86, 92, 199, 200, 228, 449, 834 PSRECFIELD 21, 199, 200, 228, 238, 269, 291, 448, 829, 834 **PSRELEASE 449** PSSQR Shell 710 PSSQR1 629 PSSQR2 629 PSSQR3 629 PSSQR4 629 PTCSQLRT 721, 729 PTCUSTAT 749 PTPEMAIN 769, 771, 788, 804, 944, 946, 952 PTPSQLRT 717,719,720,721, 763 PTPUSTAT 749, 750, 755, 756 push button 23, 154, 160, 161, 164, 183, 241, 242, 508, 512

#### Q

Query Access Trees 66 Query Security record 91 Query tool 662, 665 Query views 84 Queued (Process Status) 579, 639

# R

Radio Button 460 read-only access 58 Reasonable Date 238 Rebuild SQL Statements 781 RECNAME 833 record 84, 86 level one 342 level zero 341 Record audit 92 Record definition 205, 255, 497 Record Display Icons 77 record events 261 record field 93, 102, 269 Record Field properties 41 Record Field references 273

INDEX

Record PeopleCode 261 Record tab 103 Record Type 92 RecordChanged (PeopleCode function) 382 RecordDeleted (PeopleCode function) 383 RecordName 274 RecordNew (PeopleCode function) 383 records delete 519 link 342 location header 346 primary 344 recurrence 686 definition 702 Recurrence Definition panel 577 Recurrence Name 623 Related Display 105, 229 field 501 Related Language record 91 removing items from arrays 423 Repeat 284 REPLACE\_ALL 36 **REPLACE\_DATA 37** Report Filter 84 report output 582 Report Request Parameters 574 Request Parameters 579, 580, 655,752 Required Field 238 Restart 781, 946 retrieve data 230 Return (PeopleCode statement) 396 return code (COBOL) 728 return value (PeopleCode) 396 Rich Text File (.rft) 708 ROLLBACK 729 Round (PeopleCode function) 380 row by row processing 773 Row Select processing 232

RowDelete (PeopleCode event) 26, 236, 239, 262, 303, 304, 327, 354, 355 RowFlush (PeopleCode function) 367 RowInit (PeopleCode event) 26, 261, 296, 297, 302, 304, 354, 556 RowInsert (PeopleCode event) 26, 236, 239, 262, 303, 354, 383 Row-Level security 660, 662, 665.668 RowScrollSelect (PeopleCode function) 363 RowScrollSelectNew (People-Code function) 365 ROWSECCLASS 127, 128 RowSelect (PeopleCode event) 26, 232, 262, 301, 315, 324 Rowset (PeopleTools 8) 420 RTrim (PeopleCode function) 376, 386 Run Control 574, 887–941 Run Control ID 574, 621, 635 Run Control panel 574, 605, 633, 684, 893 Run Control record 583, 600, 633.672 Run Location 576, 623 Run Recurrence 699 Run with Defaults 16 running multiple requests 883 runtime 245 runtime errors 261

# S

save 243, 307 save events 386 Save functions 385 save processing 243, 307 SaveEdit (PeopleCode event) 26, 262, 299, 307, 327, 380 SavePostChg (PeopleCode event) 262, 309, 315, 322, 324, 334 SavePreChg (PeopleCode event) 26, 262, 308, 315, 324, 334 saving panel 106 schedule parameters 686 ScheduleProcess (PeopleCode function) 388, 394 Scheduler parameter list 646 scheduling, recurring basis 684 schema 85 Script File Options 214 scroll area 303, 306 scroll bar 166, 168, 169, 171, 172, 173, 174, 177, 342, 482 scroll bar objects 341 scroll functions 353 enhanced 420 scroll handling 257 scroll levels 344 ScrollFlush (PeopleSoft function) 351, 354 ScrollPath 346 scrolls 340-373 ScrollSelect (PeopleSoft function) 347, 354 ScrollSelectNew (PeopleSoft function) 350 search fields 220 search in PeopleCode 411 Search key 94, 132, 145, 152, 553 search processing 218, 295, 300 add mode 295 search record 144, 148, 151, 218, 533, 534, 612 SearchInit (PeopleCode event) 26, 262, 296, 297, 300 SearchSave (PeopleCode event) 26, 262, 300, 301 Secondary panel 23, 179, 182, 183 SECTION 777

section reusability 846 security 119, 129, 901 database 660 field-level 660 implementing 659-682 Row-Level 660 row-level 668 security access 488, 523, 546, 615 Security Administrator 1, 32, 47, 118, 119, 123, 126, 218 security class 61 security in SQR 672 security layers 659 security record 537 security search views 664 SELECT 337, 730 Select 731, 777, 785, 811, 833 Select Data 725, 745 Select Setup 724, 741, 745 Select statement 92 selection logic 591 Server Agent 580 Server Name 623 Set Control field 91, 98 SET INPUT 36 SET OUTPUT 35 set processing 772, 775 SetCursorPos (PeopleCode function) 307, 327 SetDefault 310 SETENV.sqc 635 SETID 98 SetNextPanel (PeopleCode function) 387 SetTracePC (PeopleCode function) 414 Setup 595 setup list 724, 742 Signed Number (Field Attributes) 80 signing-on connection type 10 database name 11 operator ID 11 password 11

Sign-On 121 sign-on times 51, 53 SQC file 635, 647 SQL Alter 212 SQL definition 254, 418 SQL in COBOL programs 716 SQL object 418 SQL statement 763, 784, 800 SQL statement name 723 SQL statement table 718 SQL string 334, 337 SQL table name 92 SQL tables 198, 201, 203, 204, 205, 209, 226, 261, 302 SQL Trace 758, 765 SQL View 84, 92, 185, 202, 205, 214, 226, 302, 496 SQL-CURSOR-COMMON 723, 732 SQL-Error 639 SQLExec function 272, 274, 276, 277, 295, 309, 332, 333, 405, 521, 786 SQLRT 722 SQL-STMT 723, 731 SQLTABLENAME 21 SQR 257, 283, 569, 571, 771, 772, 797, 817, 829, 850 command line 572 Report 621 SQR dialog box 636 SQR process 621 SQR program 572, 631 designing 591 executing 597 output files 597 SQR Runtime 624 SQR security 672 SQR security flag 669 SQRW dialog box 666 SOT 624 stamping the database 453 Standard menus 113 Startup tab 12 State Record 957 STATEMENTS 777

statements 278 STATIC 814, 822, 827, 1056 Static Image 22 Static Text 22, 102 status 449, 639 absent 449 In Progress 310 Unknown 449 STDAPI.sqc 635, 637, 638, 653 Stdapi-Init (SQC procedure) 636, 637, 639, 653 Stdapi-Term 636, 638, 639, 653 STEP 777 STMT\_NAME 719,723 STMT TEXT 719 STMT TYPE 718, 723 StopFetching (PeopleCode function) 301 Store command 719 stored SQL statement 718, 727 STRING (Data Type) 272 string functions 377, 386 structured programming 717 Structured Query Report Writer (See SQR) style 106, 315, 324 style parameter 316, 319, 324 style sheet 255 subpanel 179, 180, 181, 182, 460, 464, 466 subrecord 84, 180, 181 Substring function 386 Success (Process Status) 579, 639 Summary Trees 66 Suppress Error 784 syntax check (PeopleCode) 326 SYSADM 667 system edits 37 system tables 1002-1007 system variables 276

# Т

target orientation 447 testing 523–524, 547, 563 Text Files (.txt) 708

INDEX

Think Time PeopleCode 295 Third Party panel 669 three-tier 4, 247 Through Search field 94 Through Search key 149 TIME (Data Type) 272 Time (PeopleCode function) 377 Time-Out 49 TimePart (PeopleCode function) 272 toolbar 16 toolbar icon 265 tools tables 289 TotalRowCount (PeopleCode function) 372 trace 413 trace file 759, 763, 787, 937 contents 763 sample 943 trace filename 528 trace options 530, 780 trace settings 759 Trace SQL statement 530 Trace utility 528 Traffic Light button 575 trailing characters, removing 386 TransferPanel (PeopleCode function) 388 translate 140, 141, 144 Translate Table edit 97, 238 translate values 42, 140 Tree Manager 1, 32, 64 Trees 23, 128 True 279 Turbo 348 Turbo parameters 347 tutorial 789

Tuxedo 5, 6, 247 two-tier architecture 4

#### U

UnGray (PeopleCode function) 385 UnHide (PeopleCode function) 384 Unicode 711 UPDATE 336, 337, 732 Update 734, 777, 785 Update mode 300 Update/Display 16, 153, 176, 178, 221, 224 Update/Display All 16, 153, 176, 178, 225 UpdateValue (PeopleCode function) 370 upgrade 27, 438, 563 Upgrade Compare process 84, 444, 448 upgrade considerations 433 Upgrade Copy 84 Upgrade Copy process 444 Upgrade tab 443, 564 Upgrade values 449 Upper (PeopleCode function) 386 Use tab 104, 106, 111, 117 User Think-Time 322 USER AET 817 using Select method 421 utilizing MessageBox 320

# V

validate fields 298 vanilla 429 variables bind 334 global 278 inline bind 334, 336, 339 output 335 temporary 272 verify mode 225 view definition 497

#### W

Warning (PeopleCode function) 298, 304, 307, 309, 320, 327 web server 10 When 281 WHERE 35 While 285, 286 WinBatch 764 Windows Notepad 582 Windows Registry 759, 760 Windows Registry Editor 760 WinMessage (PeopleCode function) 324, 400 Work record 302 creating 506 work scroll 173 WorkFlow (PeopleCode event) 27, 262, 308, 315, 324.334 working storage 726, 731, 740 worklist 277

# Х

XLATTABLE 140, 167

# Υ

Yes/No Table edit 97, 238

# Essential Guide to **PEOPLESOFT** Development and Customization

Tony DeLia 🔸 Galina Landres 🔸 Isidor Rivera 🔸 Prakash Sankaran

PeopleSoft is a collection of software modules for common business functions. It is also a toolset for companies to modify, customize, and enhance the modules to suit their needs. Essential Guide to PeopleSoft Development and Customization is a comprehensive reference and tutorial that covers PeopleSoft 7.5 and gives an overview of new features in release 8.0.

Both novice and experienced programmers will benefit from this book's coverage of all aspects and elements of PeopleSoft application development. Topics run from the basics of the PeopleSoft development paradigm to the secrets of such powerful tools as SQR, Application Designer, PeopleCode, PeopleSoft COBOL, and the ins and outs of PeopleSoft customization. One of the most powerful but least understood PeopleSoft tools, the Application Engine, gets special coverage, including dynamic SQL, decision logic, and dynamic sections.

This book's broad range and detailed, practical advice will give you the skills necessary to compete in the PeopleSoft marketplace for years to come.

# What's inside

- Basics of application development
- PeopleCode
- PeopleTools
- Structured Query Report Writer (SQR)
- PeopleSoft for on-line applications
- PeopleSoft customization with real examples
- The Application Engine and Application Designer
- PeopleSoft COBOL and the PTPSQLRT module

Together, Tony Delia, Galina Landres, Isidor Rivera and Prakash Sankaran, have a very large amount of experience implementing PeopleSoft applications. This book is the end result of knowledge they accumulated in the past ten years since the first release of PeopleSoft.



\$64.95 US/\$94.95 Canada

"There's a vital need for this book to help answer our questions and aid us on the job."

—Steven Britt Tropical Shipping, LTD

"... it deals with development pitfalls head-on so you know how to deal with them—or avoid them altogether."

> —Peter Choi PeopleSoft/Oracle DBA APG Solutions & Technologies

"I will definitely recommend the book for those new to PeopleSoft. It is also a great reference for experienced developers."

—Cary Cloud Technology Solutions Company

www.manning.com/delia

