



Essential Guide to

PEOPLESOFT

Development and Customization

Module Two

- In Depth Programming in PeopleCode
- Using Application Engine in PeopleSoft
- PeopleTools Development
- Application Designer Techniques

Tony DeLia
Galina Landres
Isidor Rivera
Prakash Sankaran

For electronic browsing and ordering of this and other Manning books, visit <http://www.manning.com>. The publisher offers discounts on this book when ordered in quantity. For more information, please contact:

Special Sales Department
Manning Publications Co.
32 Lafayette Place Fax: (203) 661-9018
Greenwich, CT 06830 email: orders@manning.com

©2001 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

- © Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end.



Manning Publications Co.
32 Lafayette Place
Greenwich, CT 06830

Copyeditor: Adrienne Harun
Typesetter: Dottie Marsico
Cover designer: Leslie Haimes

Printed in the United States of America
1 2 3 4 5 6 7 8 9 10 – VH – 03 02 01 00

Using SQR in PeopleSoft applications

Structured Query Report Writer (SQR) is a programming language currently owned and supported by BRIO Technology. SQR, one of many third-party tools that comes packaged with PeopleSoft, combines the power of SQL, the sophistication of procedural logic, and the freedom of multiple-platform development. PeopleSoft has used SQR extensively for many of their batch, reporting, and upgrade applications. Because PeopleSoft was deliberately designed to perform against a variety of operating systems and databases, it is no surprise the makers of PeopleSoft chose SQR for many of their programming requirements. SQR is a powerful and flexible language that can run on multiple platforms and supports almost all relational databases. Operating systems include Windows, DOS, Unix, and MVS. Some examples of supported databases are Oracle, DB2, SQLBase, and Informix.

Contrary to what its name may suggest, SQR is much more than a reporting tool. Using SQR we can write sophisticated and robust applications that perform data manipulation, file handling, data extraction, data loading, and, of course, reporting. *SQR in PeopleSoft and Other Applications* (Landres, Galina and Vlad Landres. Greenwich, CT: Manning Publications, 1999.) is a comprehensive guide that covers all aspects of SQR development. The authors summed up SQR quite succinctly in their introductory section when they proclaimed: “(SQR) is a serious tool for serious people.” Once you have used SQR for even a short period of time, you will no doubt agree with them. The chapters ahead pick up where Galina and Vlad Landres left off in their book. More emphasis is placed here on integrating SQR with PeopleSoft components such as Process

Scheduler, Process Monitor, and Run Controls. We further enhance our Problem Tracking application by creating custom SQR reports that are executed via Process Scheduler. The reader can explore the full SQR development cycle, which includes creating the Run Control record, panel, and panelgroup; adding the panel to a menu; assigning operator security to the new menu item; and creating the process definition for the SQR program. In addition, we cover topics such as implementing security in SQR and scheduling recurring jobs. We end the section with an overview of new SQR and Process Scheduler features in release 8.0.



CHAPTER 25

Running SQR programs in PeopleSoft applications

- | | |
|--|--|
| 25.1 How SQR programs run under PeopleSoft 572 | 25.5 Viewing the status of your report via the Process Monitor 578 |
| 25.2 Selecting a report from a menu 573 | 25.6 Viewing the report output 582 |
| 25.3 Using the Run Control 574 | 25.7 Editing Run Control records 583 |
| 25.4 The Process Scheduler Request dialog 576 | |

PeopleSoft applications offer a wide range of query and reporting tools which enable users to access the necessary information for both day-to-day and long-term business decisions. For each product (HRMS, Payroll, Financials, and so forth), PeopleSoft delivers a set of standard canned reports as a part of its basic package. At the same time, PeopleSoft offers a number of tools designed to help developers customize existing reports as well as create new ones.

PeopleSoft has selected Structured Query Report Writer (SQR) as one of its main reporting and processing tools because SQR provides a flexible and robust report-writing environment. SQR works beautifully when your report needs complex procedural logic or tricky database manipulation; when you need to run your report on multiple platforms; when your report structure is complex with multiple breaks; or when you need to combine data base retrieval with special row processing.

SQR programs are not distributed in the form of platform-dependent executables. They can be easily moved between platforms, either at source level or as pre-compiled pseudo-code modules. All SQR commands, directives, and operators are platform-transparent and require no changes when the programs are moved across platforms. At the same time, programmers are free to invoke any operating system's specific commands or utilities if they feel the benefits of platform independence are outweighed by other considerations such as performance, ease of maintenance, or the need to integrate their programs into certain specific environments.

25.1 *HOW SQR PROGRAMS RUN UNDER PEOPLESOFT*

Let's start by discussing the way PeopleSoft interacts with SQR programs at a conceptual level without going into many details.

In most cases, PeopleSoft users initiate their requests for reports via PeopleSoft online panels. These panels can be delivered by PeopleSoft or developed by application programmers.

When online panel information is filled in, PeopleSoft generates process request parameters. These parameters usually include the operator ID, Run Control ID, run location, output destination, file/printer name, plus application-specific parameters, for example, Company ID, From Date, To Date, and so on.

After the process request parameters have been read from an online panel, PeopleSoft passes them to the Process Scheduler.

The Process Scheduler is a tool that enables users of the PeopleSoft system to manage PeopleSoft batch processes. Any program that runs under the PeopleSoft Process Scheduler is called a process. It can be a reporting program, a file generation program, an interface to another system, a database update program, and such. You can run processes on your workstation or on the server. You can also combine several processes into job streams and schedule them for a subsequent execution. (We will discuss this later in chapter 30.)

The Process Scheduler generates the SQR command line with flags and arguments required to run the requested SQR program, invokes SQR, and passes the flags and arguments to SQR. When the input from the online panel is saved, the system updates a number of tables that are used by SQR to communicate with the Process Scheduler and the Process Monitor via the special PeopleSoft API.

The requested SQR program is executed. It may generate reports, update the database, create flat files, or print its reports directly on the specified printer. Users are kept informed about the program status with the help of the PeopleSoft Process Monitor. The Process Monitor receives the program feedback via the PeopleSoft API parameters and displays the program status on the Process Monitor panel.

25.2 SELECTING A REPORT FROM A MENU

PeopleSoft-delivered reports are usually displayed under the Report or Process menu bar item. In most cases, programs that generate output for printing or displaying are listed under Report, while programs that manipulate the database records or generate flat files are listed under Process. Many programs do several jobs: print reports, update the databases, and generate files.

To run a report, select it from the appropriate menu: the Report menu or the Process menu. First, you have to know, of course, which report you need to execute. Let's take a look at all available reports within the Administer Workforce U.S. menu.

To begin, let's open for example, the Administer Workforce U.S. menu panel, click on the Report menu bar, and select the Years of Service report to run (figure 25.1).



Figure 25.1 Selecting a report to run under the Process Scheduler

The system asks you to choose between the Add and Update/Display options. The Add or Update action does not indicate adding or updating a report; it means adding or updating a special Run Control ID associated with each process request.

25.3 USING THE RUN CONTROL

When we run a process via the Process Scheduler, we need to supply it with a number of parameters such as Run Location, Output Destination, File/Printer name, and so on. This information is stored in the PeopleTools Run Control record PSPRCSRUNCNTL.

In addition, each process maintains its own application Run Control record to store the process-specific input run-time parameters, for example, As-Of-Date, Company Code, or State. The Run Control ID along with the Operator ID are the key fields in both application Run Control records and PeopleTools Run Control records.

Let's get back to our previous panel (figure 25.1). In this panel, the system asks you to create a new PeopleTools Run Control record or select an existing one. If an existing record fits your process execution requirements, you can select the Update/Display option, find the proper record, and reuse it. Otherwise, you need to select the Add action and add a new Run Control ID for a new record. We create a new Run Control by selecting the Add action (figure 25.2).



Figure 25.2
Adding a new Run Control record

We enter MY_RUN01 on the system's prompt. After pressing the ENTER key or OK button, the system displays the next screen called the Run Control panel.


What you see in figure 25.3 is the Run Control panel for the Years of Service report. The Operator ID value is your PeopleSoft logon value. The Run Control ID is the value you entered in the previous panel.

Please note that this panel includes a process-specific portion called "Report Request Parameters," which may look different for each report. This part of the panel contains all input parameters necessary to run the report. In our case, there are two parameters: As Of Date and Years of Service. Let's enter 01/01/2000 as the value for the first parameter and 10 for the second one. The upper portion of the panel is a standard subpanel usually included in most Run Control panels. It contains the Operator ID, the Run Control ID, and the Language. A similar subpanel without the Language is also available.

As soon as this panel is saved, an application Run Control record is created. This record, identified by the Run Control ID and the Operator ID, stores the report input parameters. It will allow you to reuse these parameters in the next report runs. The next time you need to run this report, you simply select the proper Run Control ID (in our case, it is MY_RUN01), and the system automatically retrieves the settings. (Note that some HRMS applications may delete their application Run Control records upon successful execution).



Figure 25.3 The Run Control panel for the Years of Service report

In order to run the report, click on the Traffic Light  button or select File, Run. You will go to the next panel, which displays the Process Scheduler Request dialog panel, where you can specify when and where to run the report (figure 25.4).

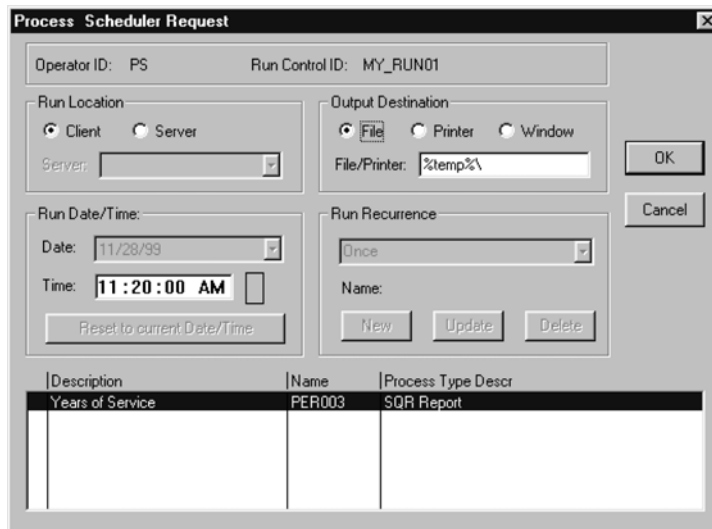


Figure 25.4
The Process Scheduler Request dialog panel

After you select all the settings on the Process Scheduler Request dialog panel, the PeopleTools Run Control record (PSPRCSRUNCNTL) is updated. This gives the PeopleSoft Process Scheduler the necessary information to run and monitor the process request.

25.4 THE PROCESS SCHEDULER REQUEST DIALOG

The Process Scheduler Request dialog panel (shown in figure 25.4) is used to specify where you want to run your report, the destination of your report's output, and the time of the actual run. Let's take a closer look at these parameters.

The Run Location group box allows you to choose between running your report on *Client* or *Server*. If you select *Server*, you have to select the server name from the list of available servers. If you want to schedule your process to run at a later time, you must select *Server* because your process scheduling can only be done on a server, and not on your client machine.

TIP Always select a server name when Run Location is *Server*.

The Output Destination of your report may be *File* or *Printer*. The Window output is available only for Crystal Report programs, not for SQR programs. If you want to direct your output to a file, you need to enter the filename and a complete path for the file in the Printer/File text box. If you enter an existing file name, the system will overlay the old file.

If you need to print a hard copy of the report, select *Printer* and specify which printer port to use in the Printer/File text box.

The Run Recurrence and Run Date/Time parameters are only available if you select *Server* as the run location. The Run Recurrence parameter allows you to define your process as a recurring process that may be executed on a periodic basis.

Run Date/Time defaults to the current date and time. If you plan to schedule only one report run, select the *Once* option in the Run Recurrence selection and enter the desired run date and time in the Date and Time boxes. If you select a Run Recurrence value other than the default value *Once*, and specify the proper run recurrence definition, it will override any Run Date/Time you may have previously set.

Let's see how you arrange to run your process at a specific time. Remember, you can use this option only by specifying the run location as *Server*. If, for example, you want to run your report every Sunday, weekly at 7 AM, you have to create a new Run Recurrence definition (if one has not been set up already): click on the *New* button and name it *WEEKLY AT 7 AM* (figure 25.5).

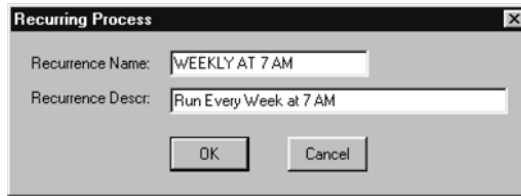


Figure 25.5
creating a new Run Recurrence
definition

When you press the OK button, the Recurrence Definition panel appears (figure 25.6). This panel allows you to define the starting date and time, the run frequency, and all other necessary scheduling information.

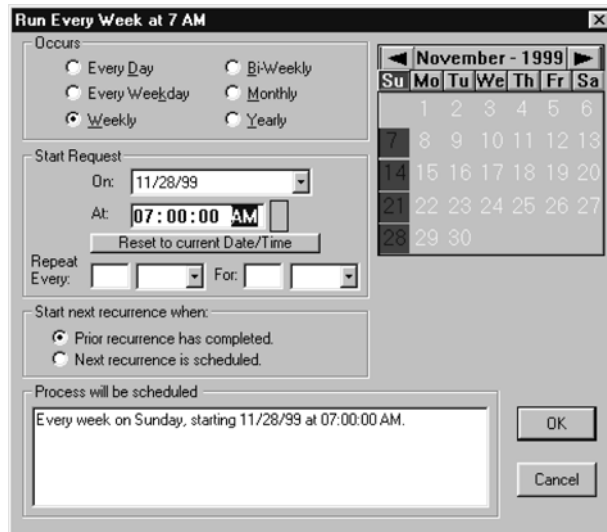


Figure 25.6
The Recurrence Definition
panel

It is important to keep in mind that all the report-run schedule information is entered for a specific Run Control record, identified by a combination of the Run Control ID and operator ID. You should not use this Run Control ID to run or schedule other processes. Technically, the system will allow you to do this, but all your previous settings will then be overlaid with the new settings.

TIP Use meaningful Run Control IDs.

For example, the Years of Service report may be scheduled to run every Sunday using a Run Control ID named `Sunday_Run`. If, in addition to this run schedule, you want to run the report at the end of the month, another Run Control ID must be used; otherwise, the every Sunday run schedule will be overlaid.

Now that you know how to schedule report runs, let's go back to figure 25.4 and review the run parameters on the Process Scheduler Request Dialog panel for the Years of Service report. We select the Run Location as `Client` and the Output Destination as `File`. We need to make sure that the output file goes to the proper directory. Since we run the report on the client machine and cannot schedule it to run at a specified time, the run time defaults to the current time. The only thing you need to do is to press OK. Another PeopleSoft tool, called Process Monitor, can help you monitor your process. (Remember, every report run under the Process Scheduler is a *process*!)

25.5 VIEWING THE STATUS OF YOUR REPORT VIA THE PROCESS MONITOR

The Process Monitor not only allows you to check the status of your process, but also permits you to see the report run parameters, delete the report from input or output queue, and perform other tasks (figure 25.7).

Navigation: Go → PeopleTools → Process Monitor

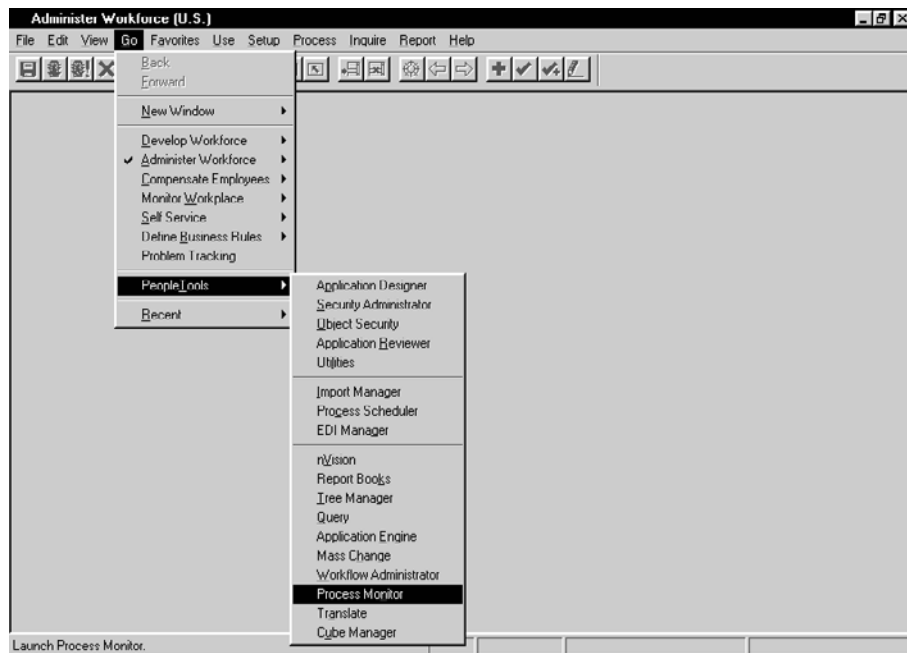


Figure 25.7 Invoking the Process Monitor

After selecting the Process Monitor, the system displays the Process Monitor panel (figure 25.8).

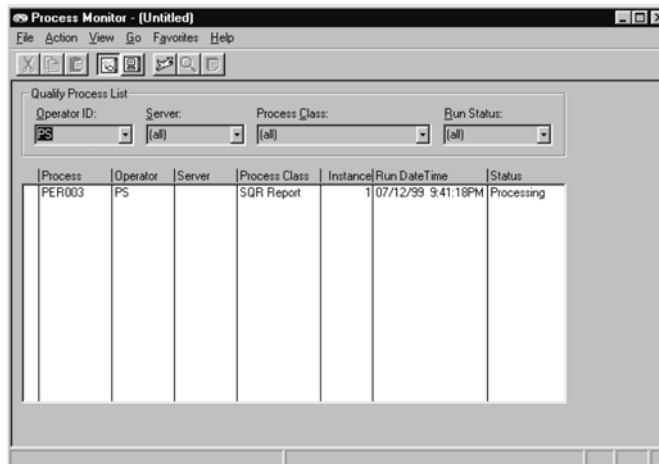


Figure 25.8
The Process Monitor
panel with Process
Status = Process

The Process Monitor panel in figure 25.8 displays the information about all processes by the operator ID. You can easily modify the view by narrowing the selection down to a specific operator ID, server, process class, and run status. The last column in this panel (named “Status”) shows the status of your process. The status could be one of the following: Success, Initiated, Hold, Queued, Processing, Canceled, Error.

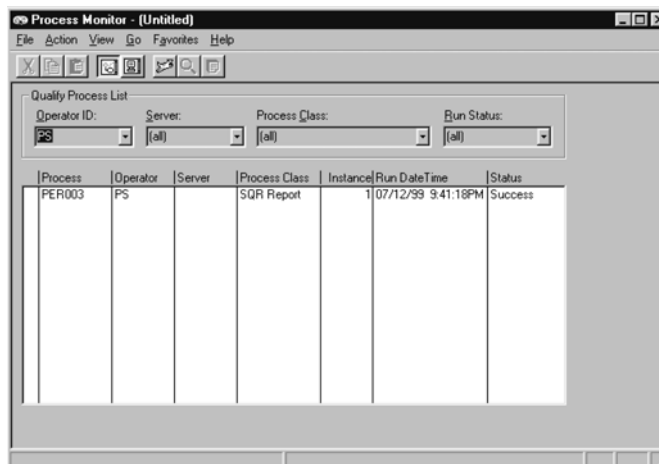


Figure 25.9
The Process Monitor
panel with Process
Status = Success

Let’s double-click on your process. The system displays the Process Request Detail panel group, which consists of two panels: Process Detail and Request Parameters (figure 25.10).

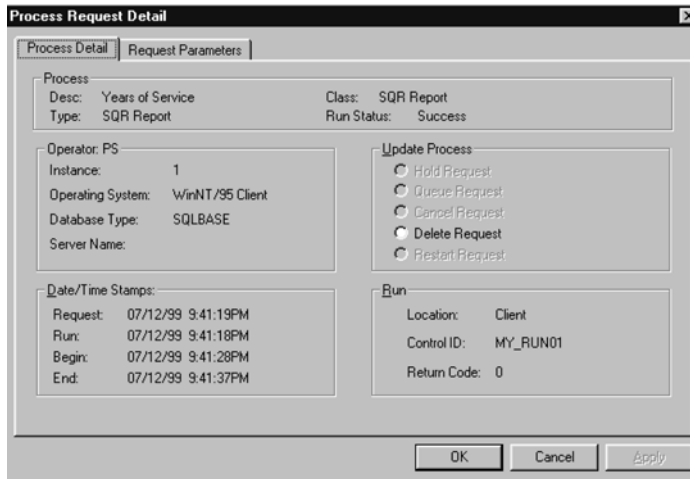



Figure 25.10
The Process Detail
panel

The Process Detail panel shows the process information including the process description, type, and run status, the ID of the operator who initiated the process; the operating system under which the process is run; the database type; and the server name if the process is run on a server. You can check to see how long your report took to run by looking at the beginning and ending date/time stamps. Knowing these details can be useful in troubleshooting. Sometimes, if the process you started just sits in the input queue with its status equal to Initiated or Queued, and you wonder what's going on, the first thing to do is check the Process Detail panel to see whether the process was initiated on the client or the server. If the process was initiated on a server, was the correct server name entered? You can check to see if the appropriate Server Agent is up and running by selecting View, Servers or by clicking on  in the main Process Monitor panel. If you need more information, go to the second Process Request Detail sub panel and examine the request parameters (figure 25.11).

This panel displays all parameters passed to your process, as well as the command line used during the execution. You can also use the Copy to Clipboard push button if you need to save and review your process request parameters.

TIP If you forget the destination of your output report file, you can always check it in the Request Parameters panel. The output destination will be shown in the Parameter list following the `-f` flag.

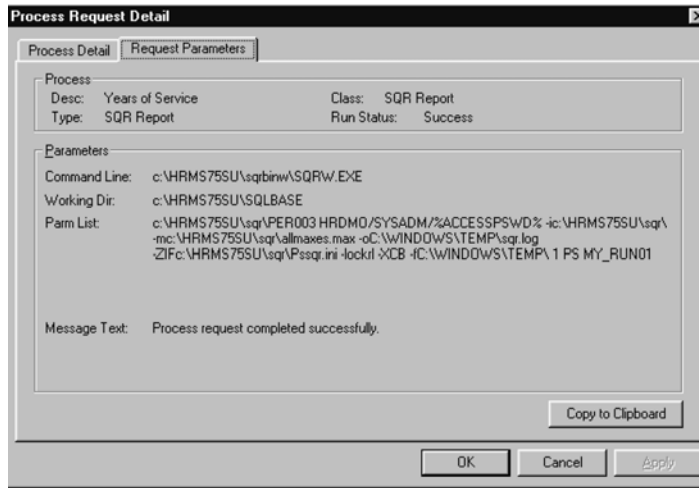


Figure 25.11
The Request
Parameters panel

25.5.1 Controlling your processes via the Process Monitor

The Process Monitor panel gives you complete control over your process. You can cancel, delete, or put the process on hold. Depending on the current status of your process, the system will allow you to select a valid action. For example, if the process has finished, you can delete it from the Process Monitor panel. If the process was just initiated, you can cancel the process, and then delete the process status record from the panel (figure 25.12).

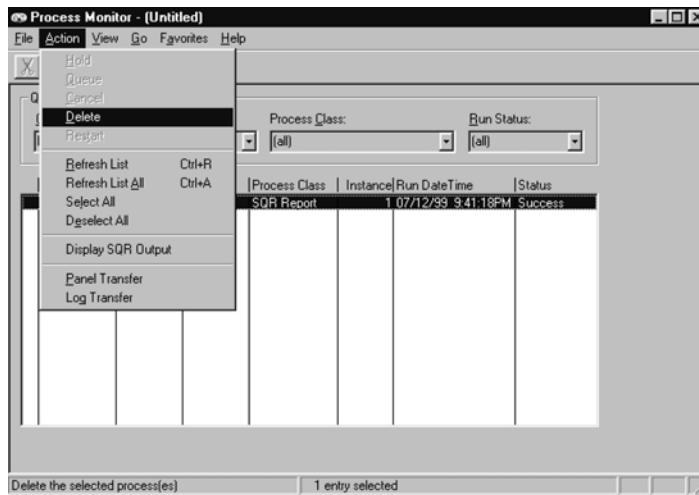


Figure 25.12 Deleting the process status record from the Process
Monitor panel

25.6 VIEWING THE REPORT OUTPUT

When you run your report on the client, you can display your report output from the Process Monitor panel. Click on your process, then select Action, and Display SQR Output (figure 25.13).

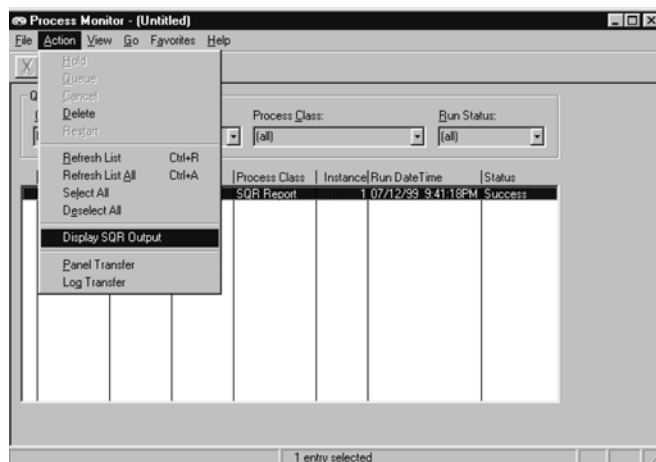


Figure 25.13
Displaying SQR output

You can see the output of your program displayed on Windows Notepad (or WordPad if the output is too large). Note that, if the output is in the .lis format, you see the special print control characters on the first line of your report, (figure 25.14). When you print the report on a printer, the report comes out without the special characters in the format specified by your program.

		Years Of Service		
		-Service-		
Employee Name	Hire Date	Yrs	Mths	Department
Jordan, Robert	09/26/1963	36	3.2	Office of the Preside
Kidd, Kenneth	12/12/1966	33	0.7	Office of the Preside
Drew, Suzanne	07/06/1976	23	5.9	Office of the Preside
Bell, Anna	09/21/1977	22	3.4	Office of the Preside
Smith, John	01/12/1980	19	11.7	Office of the Preside
Fletcher, Leslie	03/01/1980	19	10.0	Office of the Preside
Cone, Alton	06/09/1980	19	6.8	Office of the Preside
Albright, Arnold	01/01/1981	19	0.0	Connecticut Operator
Avery, Joan	04/16/1981	18	8.5	Human Resources
Koontz, Stephan	04/16/1981	18	8.5	Controller (MOB)
Schunacher, Simon	04/16/1981	18	8.5	Office of the Preside
Marks, Michael	08/15/1981	18	4.6	Field Research & Deve
Stevens, Susan	08/15/1981	18	4.6	Field Research & Deve
Frumman, Wolfgang	01/01/1982	18	0.0	Business Services
Limburg, James	01/01/1982	18	0.0	Business Services (MC
Penrose, Steven	01/01/1982	18	0.0	Business Services
Vincent, Catherine	01/01/1982	18	0.0	Illinois Operations
Andrews, Frank	01/21/1982	17	11.4	Human Resources
Jones, Theresa	06/01/1982	17	7.0	Operations Administra
Kean, Betsy	06/01/1982	17	7.0	Business Services (MC
Sullivan, Theresa	06/01/1982	17	7.0	Operations Administra
Walters, Gary	06/01/1982	17	7.0	New York Operations
Elias, Jan	10/03/1982	17	3.0	Office of the Preside

Figure 25.14 SQR output displayed via the Process Monitor panel

If you run your report on the server, you cannot see the report output from the Process Monitor panel. You can use FTP or another available tool to copy your report output from the server to the client. Another option is to print your report directly from the server to your network printer.

25.7 EDITING RUN CONTROL RECORDS

You already know that any process execution is controlled by two types of records: the PeopleTools Run Control record and the application Run Control record. Both records have the same key identifier, a combination of the Run Control ID and the operator ID.

The PeopleTools Run Control record stores the generic report control information: where to run the report and where to direct the report output. Additional information related to the process run request specifics, is stored in the Process Request system table.

In order to see the PeopleTools Run Control record, select Edit/Preference/Run Control from the Administer Workforce (U.S.) panel (figure 25.15).



Figure 25.15 Selecting the Run Control Edit panel



Figure 25.16 Edit Run Controls dialog

The system displays the Edit Run Controls dialog window shown in figure 25.16.

You can see all the Run Control record IDs that you are allowed to access based on your security, including the one you just created: MY_RUN01. Select this ID and press the OK button to see the record details. The system brings the Edit Run Control panel shown in figure 25.17.

As you can see, the predefined settings for this Run Control ID are displayed on the panel. The panel shows the run location as `Client` and the output directed to `File`.



Figure 25.17 The Edit Run Control panel



Figure 25.18 Changing the Run Control settings

order to see this record on the Administer Workforce (U.S.) window (figure 25.19), select Report/Years Of Service and, this time, select the Update/Display option.

Press the CANCEL button and return to the previous window (figure 25.16). As you can see from the dialog panel, you have the ability to create new Run Control records, edit existing records, or even delete records you no longer need. Keep in mind that, if you delete a Run Control record, only the PeopleTools Run Control record will be deleted. All Application Run Control records with the same key values remain in the system. These records can be deleted via a database management tool outside of PeopleSoft. Depending on your database, you can use products like SQL*Plus for Oracle, SQL Programmer for Sybase, QMF for DB2, or similar ones.

Let's see what would happen if you changed some of the Run Control parameters. On the Edit Run Controls dialog (figure 25.16), select MY_RUN01 again and press the EDIT button.

The system brings back the Edit Run Control panel with all the previously specified settings. Now change the Run Output setting from File to Printer and press the OK button to save the changes (figure 25.18).

Now that we have figured out how to see and change the PeopleTools Run Control records, let's see if you can do the same with the application Run Control records.

The application Run Control record contains all application-specific report input parameters. In

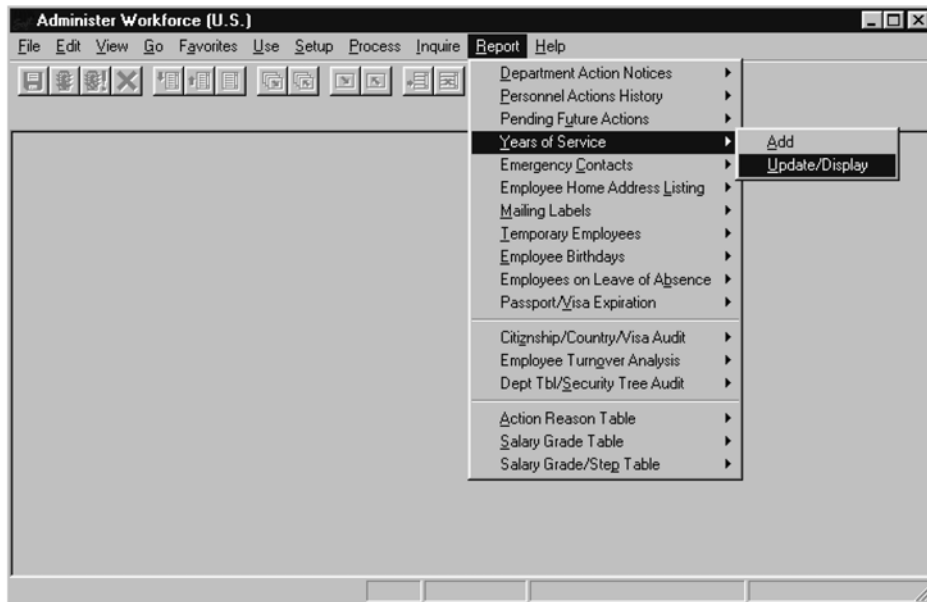


Figure 25.19 Displaying an existing application Run Control record

After you press the OK button, the system displays the list of all available Run Control IDs (figure 25.20).

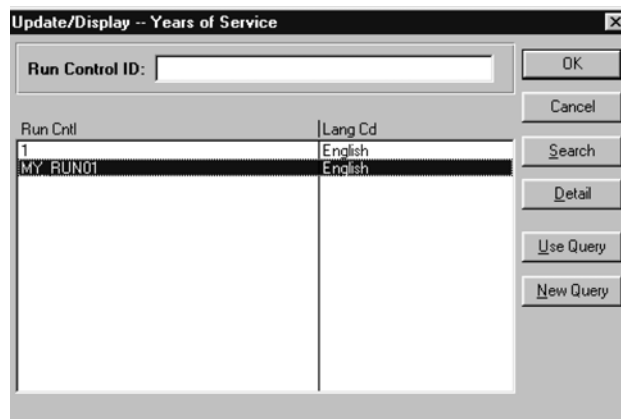


Figure 25.20
Run Control ID dialog

Select MY_RUN01, and the system displays the application Run Control panel with all process-specific parameter values (figure 25.21).



Figure 25.21 The application Run Control panel with process-specific parameters

Now you can see all previous parameter values and change some of these values if needed. In our case, there are two parameters, As Of Date and Years Of Service. Let's change the As Of Date parameter value from 01/30/1997 to 01/01/2000. Click on the Traffic Light button to run the report. The system displays the next window, the Process Scheduler Request dialog shown in figure 25.22.

As you can see in figure 25.22, the Output Destination on this panel has been changed from File to Printer. If you run the report now, the system will use both the changed PeopleTools Run Control parameters and the changed application Run Control parameters.

Operator ID: PS Run Control ID: MY_RUN01

Run Location
☒ Client ☐ Server
 Server:

Output Destination
☐ File ☒ Printer ☐ Window
 File/Printer:

Run Date/Time
 Date:
 Time:

Run Recurrence

 Name:

Description	Name	Process Type Descr
Years of Service	PER003	SQR Report

Figure 25.22 The Process Scheduler Request dialog with updated settings

KEY POINTS

- 1 PeopleSoft-delivered reports are usually executed from on-line panels and run with the help of the PeopleSoft Process Scheduler.
- 2 The Process Scheduler works with processes and job streams.
- 3 You can either run processes on your workstation or remotely on a server.
- 4 The Process Scheduler controls process executions with the help of the Run Control records: PeopleTools Run Control and the application Run Control. The Run Control ID along with the operator ID are the key fields in these records.
- 5 You can schedule a process execution at a specific date/time only if the process runs on a server.

KEY POINTS (CONTINUED)

- 6** The Process Monitor allows you to control your processes: you can view the process status or cancel, delete, or put processes on hold, depending on the status of your process.
- 7** You can view your report output if it were executed on the client via the Process Monitor panel.
- 8** The output destination of your report may go to File or Printer. Window output is available only for Crystal reports, not for SQR Reports.



CHAPTER 26

Creating a custom SQR program

- 26.1 Designing your SQR program 591
- 26.2 Executing your SQR program 597
- 26.3 Examining the SQR program output files 597

During the course of this book, we have been developing the Problem Tracking application. Our application would not be complete if we did not create reports for our users. Many reports are usually expected from Problem Tracking systems. Users may need to have a list of all open and not assigned incidents or all closed incidents listed by a particular date, by the project ID, by user, or by the person responsible to fix a problem. In this chapter, we will create a simple SQR program that will be able to support some of these functions.

We'll start with exercise 1:

Create a Problem Tracking status report.

The report will list all reported incidents sorted by Problem Status and Incident Date. As a first step, we will create an SQR program that will not be attached to any

PeopleSoft menu and, therefore, will not be available for execution from the PeopleSoft Process Scheduler. In the following chapters, you will learn how to make this program run under PeopleSoft.

26.1 DESIGNING YOUR SQR PROGRAM

Navigation: Go →PeopleTools →Application Designer →Open →MY_PROJECT



Figure 26.1 The MY_PROJECT Project

Let's take a look at the tables that we created in our Problem Tracking application and find the ones that can be used to produce our report.

As you can see from figure 26.1, the MY_PROJECT project contains all the Problem Tracking Application objects, including all the custom tables we created. Let's double-click on the MY_PROBLEM_TRKG record and examine all its fields (figure 26.2).

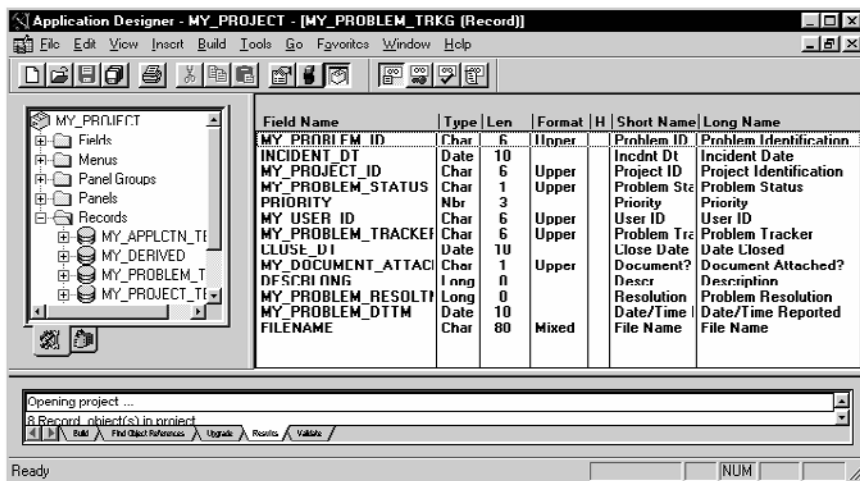


Figure 26.2 The MY_PROBLEM_TRKG record definition

When designing SQR programs, understanding your data model and building the right selection logic is a crucial part of any development process. As you can see from figure 26.2, the MY_PROBLEM_TRKG table can be used in our report since it contains all information about the incidents entered into the system via our custom online

application. In addition, our report probably needs information such as an application and a project description, a user name, and so forth. We can select this information from other tables created for this Application.

Let's create our SQR program:

Listing 26.1

MYPROB01.sqr

```
!Problem Status Report

#define problem_status_len 10
#define project_descr_len 30
#define date_len 10
#define priority_len 8
#define user_name_len 20
#define responsible_name 20
#define col_sep 2

!*****
Begin-Setup
!*****
Load-Lookup Name=Projects
    Rows = 500
        Table = PS_MY_PROJECT_TBL
        Key = MY_PROJECT_ID
        Return_Value=Descr

Load-Lookup Name=Users
    Rows = 1000
        Table = PS_MY_USER_TBL
        Key = MY_USER_ID
        Return_Value=Name

End-Setup

!*****
Begin-Heading 7
!*****
print 'Problem Status Report' (1,1) Center

page-number (0,100) 'Page No. '
print 'Run Date ' (0,100)
Print 'Problem Status: ' (0,100)
Print $Stat (0,100)
print '=' (0,100)
print 'Project Description ' (0,100)
print 'Incident ' (0,100)
print 'Priority ' (0,100)
print 'User Name ' (0,100)
print 'Responsible ' (0,100)
```

```

print 'Close          '      ( ,+{col_sep}, {date_len}      )
print '              '      (+1,          1, {project_descr_len} )
print '    Date       '      ( ,+{col_sep}, {date_len}      )
print '              '      ( ,+{col_sep}, {priority_len}    )
print '              '      ( ,+{col_sep}, {user_name_len}    )
print 'To Resolve    '      ( ,+{col_sep}, {responsible_name} )
print 'Date          '      ( ,+{col_sep}, {date_len}      )
print '='              (+1,          1,          125) fill

```

End-Heading

!*****

Begin-Program

!*****

Do Init-Report

Do Main

End-Program

!*****

Begin-Procedure Init-Report

!*****

Do Ask-Input-Parameters

Do Build-Where

Do Load-Xlats

End-Procedure

!*****

Begin-Procedure Ask-Input-Parameters

!*****

!Get User's Input

Input \$AsOfDate Type=Date 'Please enter As Of Date'

Let #Input=1

While #Input = 1

Input \$Problem_Status Type=Char 'Please Enter Problem Status
(1=Initiated, 2=Assigned, 3=Progress, 4=Testing, 5=Resolved,6=Void) or
press Enter for All' Status=#Input_Status

If \$Problem_Status = ''

Let #Input = 0

Else

If \$Problem_Status > '0' and \$Problem_Status < '7'

show 'Problem Status Entered = ' \$Problem_Status

Let #Input = 0

Else

Show 'Invalid Input, Re-Entry Required'

End-If

End-If

End-While

End-Procedure

```

!*****
Begin-Procedure Build-Where
!*****
!Build Where Clause based on user's Input
If $Problem_Status = ''
    Let $Where_status = ''
Else
    Let $Status=Rtrim($Problem_Status,' ')
    Let $Where_status = 'And A.My_Problem_Status = '|| ''''||$Status||''''
    Show $Where_status
End-If

End-Procedure

!*****
Begin-Procedure Load-Xlats
!*****
Let $Where_Xlat1 = 'FIELDNAME='MY_PRIORITY'''
    || ' and X.EFFDT = (Select max(Effdt) from XLATTABLE '
    || 'Where Fieldname=X.Fieldname And FieldValue=X.FieldValue'
    || ' And Effdt <= Sysdate and Language_Cd = 'ENG') '

Load-Lookup Name=Priority
Rows = 10
Table = 'XLATTABLE X'
Key = FIELDVALUE
Return_Value=XLATSHORTNAME
Where=$Where_Xlat1

Let $Where_Xlat2 = 'FIELDNAME='MY_PROBLEM_STATUS'''
    || ' and S.EFFDT = (Select max(Effdt) from XLATTABLE '
    || 'Where Fieldname=S.Fieldname And FieldValue=S.FieldValue'
    || ' And Effdt <= Sysdate)'

Load-Lookup Name=Status
Rows = 20
Table = 'XLATTABLE S'
Key = FIELDVALUE
Return_Value=XLATSHORTNAME
Where=$Where_Xlat2

End-Procedure

!*****
Begin-Procedure Main
!*****
Begin-Select
A.My_Problem_Status () on-break Print=Never After=Page-Break
Save=$Status_Cur
A.My_Project_ID
A.Incident_DT
A.Priority
A.My_User_ID

```



```

A.My_Problem_Tracker
A.Close_Dt
    Do Print-Line
From PS_MY_PROBLEM_TRKG A
Where A.Incident_Dt <= $AsOfDate
[$Where_status]
order by A.My_Problem_Status
End-Select
End-Procedure

!*****
Begin-Procedure Print-Line
!*****
    Lookup Projects &A.My_Project_ID $Descr
    Print $Descr                      (+1,          1, {project_descr_len}
    )
    Print &A.Incident_DT              ( ,+{col_sep}, {date_len}
    )
    Lookup Priority &A.Priority $Priority_Descr
    Print $Priority_Descr              ( ,+{col_sep}, {priority_len}
    )
    Lookup Users &A.My_User_ID $User_Name
    Print $User_Name                   ( ,+{col_sep}, {user_name_len}
    )
    Lookup Users &A.My_Problem_Tracker $Problem_Tracker_Name
    Print $Problem_Tracker_Name        ( ,+{col_sep}, {responsible_name}
    )
    Print &A.Close_Dt                 ( ,+{col_sep}, {date_len}
    )

End-Procedure

!*****
Begin-Procedure Page-Break
!*****
Lookup Status $Status_Cur $Stat
new-page
End-Procedure

!*****

```

As you can see from listing 26.1, our SQR program consists of the following sections: Setup, Heading, Program, and several Procedure sections.

In the Setup section we loaded two tables, PS_MY_PROJECT_TBL and PS_MY_USER_TBL, into the program memory with the help of the SQR Load-Lookup command. This technique is used to speed up the data lookups performed for every selected row in the Print-Line procedure.

In the Heading section we print our report header information.

In the Program section we call the Init-Report and Main procedures.

The purpose of the `Init-Report` procedure in our program is to prepare for our main reporting logic. We call the `Ask-Input-Parameters` procedure and, based on the results received from the user's input, call the `Build-Where` procedure to dynamically construct the `WHERE` clause for our main `Select`.

`Ask-Input-Parameters` interacts with users to obtain the input parameters: `As Of Date` and `Problem Status`. It also verifies a user's input and prompts again if the input is incorrect. Note that, when entering the problem status, users have an option to simply press `Enter` when they want to select records with all problem statuses.

Take a look at the `Build-Where` procedure. We are building a dynamic `WHERE` clause here. First, we check to see if our users entered any `Problem Status` or if they left this value blank (`Null`) to select all problems. If the value in the input variable is `Null`, we initialize the `$Where_status` string with `Null`. Otherwise, we build the `WHERE` clause by concatenating the column name with the status enclosed in quotes. Note that we use four quotes here. This is because we have two outside quotes to indicate the beginning and the end of a string as well as a double quote (instead of one) to tell `SQR` that this is a special character. You will see later that using this technique of building the dynamic parts of `SQL` helps us to create an efficient program.

The `Load_Xlats` procedure is designed to load lookup tables with value descriptions for the `MY_PRIORITY` and `MY_PROBLEM_STATUS` fields, thus saving on costly database operations. Why didn't we place this `Load-Lookup` into the `Setup` section along with the others? Because the `Let` statement is not allowed in the `Setup` section. Of course, other methods do exist for reading from the `Translate` table. PeopleSoft, for example, delivered a special include file `READXLAT.sqc`, which can also be used for this purpose.

The `Select` statement in the `Main` procedure selects the requested information. Its `WHERE` clause consists of two parts. The first one restricts the selection to the records with an `Incident Date` that is less than or equal to the input prompt date. The second part of the `WHERE` clause is stored in the string variable `$Where_status` that we built earlier in the `Build-Where` procedure. With this little trick, we can always select the required rows based on user's input. Another, simpler option would be to exclude `Status` variable from the `WHERE` clause and instead use the `SQR` procedural logic (`If-Then`) to check the selected rows one by one and compare the values in the column `MY_PROBLEM_STATUS` with the input variable. Our technique is clearly more efficient because we do not select unnecessary rows only to drop them later.

The `Print-Line` procedure is called from the `Main Select`. It is executed for each selected row. We perform several `Lookup` commands to get some field values from the `Lookup` arrays that we loaded into memory earlier. We then print the values. Here we use substitution variables (defined at the beginning of the program) which specify the print positions. If you need to change the layout of your report, you just have to modify these variables once where they were defined.

26.2 EXECUTING YOUR SQR PROGRAM

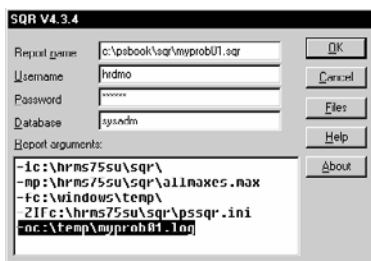


Figure 26.3 Submitting the SQR program via the SQRW dialog box

An SQR program can be invoked in different ways. You can start your program from the SQR dialog box in the Windows environment. You can also execute SQR programs from the operating system command line or call them from other programs. As an alternative, SQR programs can be run in batch mode under VAX/VMS, MVS, UNIX, MS-DOS, Windows, or OS/2 using DCL (VAX/VMS), JCL (MVS), shell scripts (UNIX), or batch files (MS-DOS, WINDOWS, OS/2).

At this point we execute our program from the SQRW dialog box. Let's fill in the SQRW dialog box as shown in figure 26.3 and submit our program.

After pressing the OK button, our program is submitted for execution. We are then prompted for our program's input parameters (figure 26.4):

```
Please enter As Of Date: 08/03/99
Please Enter Problem Status(1=Initiated, 2=Assigned, 3=Progress, 4=Testing, 5=Re
solved,6=Void) or press Enter for All:
```

Figure 26.4 Prompt for input parameters

As you can see, we entered 08/03/99 as our As Of Date parameter value and pressed Enter to select records with all statuses. When all entries are accepted, our program runs to the end.

26.3 EXAMINING THE SQR PROGRAM OUTPUT FILES

Our program created two output files: the report file myprob01.lis and the log file myprob01.log. Let's take a look at the log file first. As you may already know, the log file contains information that is displayed by the Display or Show commands from our program. In addition, it may have some system information, such as the number of records loaded for the lookup table, Input command prompt, and so on. Our log file is shown in figure 26.5.

As you can see in figure 26.5, the information in this log file is printed by the SQR engine. The information about load lookup tables is useful, especially in the testing stage since it shows you how many rows were loaded. You can easily spot a problem using this information.

```

(SQR 2613) Loading 'projects' lookup table ... done. 5 rows loaded.
(SQR 2613) Loading 'users' lookup table ... done. 7 rows loaded.
Please enter &s Of Date: 08/03/99
(SQR 2613) Loading 'priority' lookup table ... done. 3 rows loaded.
(SQR 2613) Loading 'status' lookup table ... done. 6 rows loaded.
SQR: End of Run.

```

Figure 26.5 Myprob01.sqr log file

SQR V4.3.4

Report name: c:\psbook\sqr\myprob01.sqr

Username: jhedmo

Password: *****

Database: sysadm

Report arguments: -lc:\hrms75su\sqr\ -np:\hrms75su\sqr\allmaxes.max -fc:\windows\temp\ -ZIFc:\hrms75su\sqr\pssqr.ini -oc:\temp\myprob01.log -KEEP

Figure 26.6 Creating an SPF file output along with a .lis file

Let's now take a look at our report output. The .lis file is intended to be printed and usually has formatting lines with information about the report layout, fonts, and such. You can still display the output file, but in order to test your report output, you need to print it. If you want to work with your report online, you should create an .spf file by using the `-KEEP` or `-ZIV` command line flags when submitting your program for execution. If we execute our report with the `-KEEP` flag, we would be able to see both: the .lis file and the .spf file output.

Figure 26.7 shows the output MYPROB01.lis file.

Problem Status Report Page No. 1

Problem Status: Initiated

Project Description	Incident Date	Priority	User Name	Responsible To Resolve	Close Date
HR Customizations	1999-07-01	Low	Barnie, John	Mentor, Bill	

Problem Status Report Page No. 2

Problem Status: Assigned

Project Description	Incident Date	Priority	User Name	Responsible To Resolve	Close Date
General Ledger Customizations	1998-06-01	Medium	Bitmap, Hector	Iahari, Peter	
Union Mass Change	1999-07-14	High	Iahari, Peter	Barnie, John	
Department Security	1999-07-10	High	Mentor, Bill	Kaplan, Joseph	

Figure 26.7 LIS file output

In order to see the SPF file, just double-click on C:\Windows\temp\myprob01.spf, and the SPF Viewer displays the file in a convenient online format (figure 26.8).

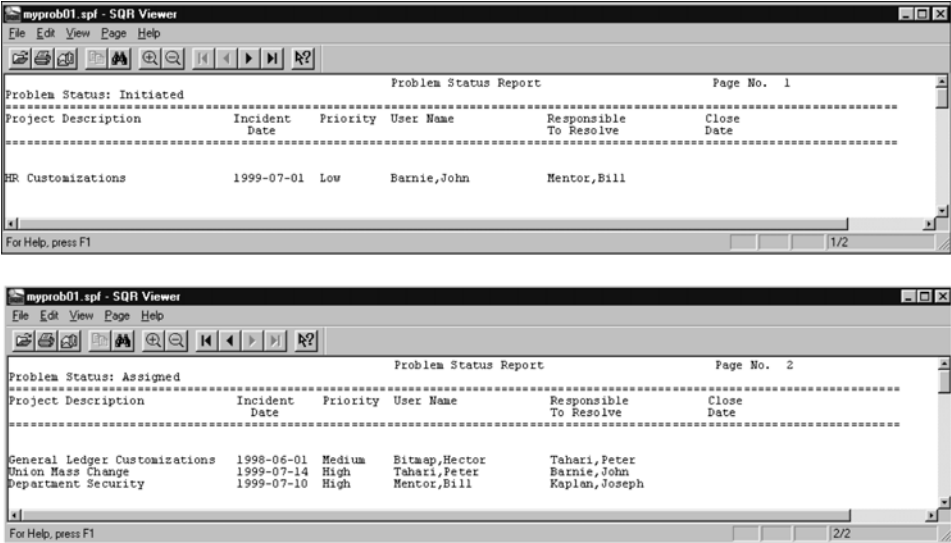


Figure 26.8 Using SQR Viewer to view SPF file output

KEY POINTS

- 1 In order to design an SQR program, you should be familiar with your database.
- 2 Dynamic SQL techniques help create more efficient programs.
- 3 An SQR program can be invoked from the SQR Dialog Box in the Windows environment or from the operating system command line. You can also call an SQR program from other programs or batch scripts.
- 4 If you would like to view your report online, you can create an .SPF file by using the -KEEP or -ZIV command line flags when submitting your program for execution.



CHAPTER 27

Attaching SQR to the Process Scheduler

27.1	Selecting a Run Control record	600	27.6	Testing your changes	617
27.2	Creating a Run Control panel	605	27.7	Creating a process definition for the problem status report	621
27.3	Creating a panel group	610	27.8	Specifying the program directory	628
27.4	Selecting a menu for your report	613	27.9	Testing your process definition	629
27.5	Granting security access	615			

27.1 SELECTING A RUN CONTROL RECORD

PeopleSoft, which delivers a number of standard reports, records, panels, and menus, has always recommended that the best way to add new functionality is to clone already developed similar application objects. We will be using this commonly accepted approach in attaching custom reports to PeopleSoft.

In PeopleSoft, the Application Run Control records are used to save the input parameters for processes. PeopleSoft developed a number of Application Run Control records that can be used if these records have the necessary fields for your program. For example, the Years of Service report uses the *As Of Date* and *Years of Service* as input parameters. Let's take a closer look at this report and find out what

Run Control record is used in the report. First, we need to find the name of the panel to which this report is attached.

Navigation: GO →Administer Workforce →Administer Workforce (U.S.) →Report →Years of Service



Figure 27.1 The Run Control panel for Years of Service report

We select View →Panel Name from the Administer Workforce (U.S.) menu, which allows us to see the panel name used for this report.

As you can see from figure 27.1, the panel name is RUNCTL_PER003.

TIP You can select most of the delivered Run Control panels by typing 'RUNCTL_' in the Application Designer. They are a good source from which to clone.

Open this panel in the Application Designer to ascertain what record is used as its Run Control record.

When the panel is opened, select Layout →Order from the Application Designer menu (figure 27.2).

Navigation: Go →PeopleTools →Application Designer →Open →Panel →
RUNCTL_PER003

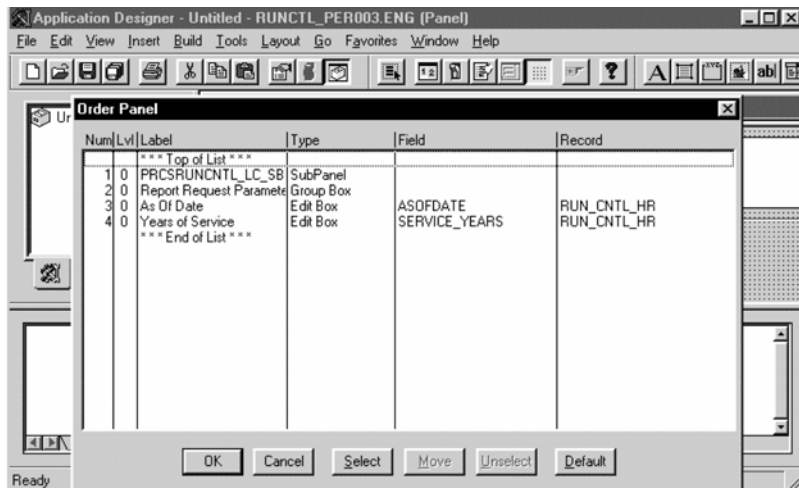


Figure 27.2 The RUN_CNTL_HR record is used as report's Run Control record

This report uses an application Run Control record named RUN_CNTL_HR. The structure of this record is shown in figure 27.3.

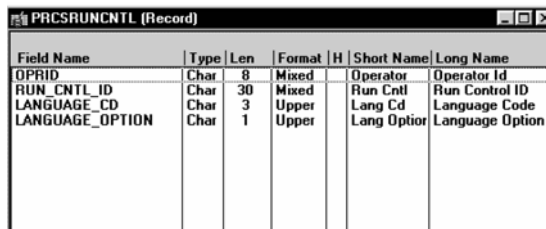
Field Name	Type	Len	Format	H	Short Name	Long Name
OPRID	Char	8	Mixed		Operator	Operator Id
RUN_CNTL_ID	Char	30	Mixed		Run Cntl	Run Control ID
FROMDATE	Date	10			From Date	From Date
THRUDATE	Date	10			Thru Date	Thru Date
ASOFDATE	Date	10			As Of	As Of Date
LANGUAGE_CD	Char	3		Upper	Lang Cd	Language Code
CALENDAR_YEAR	Nbr	4			Year	Calendar Year
SERVICE_YEARS	Nbr	2			YrsService	Years of Service
POSITION_NBR	Char	8		Num	Position	Position Number
POS_ACTIVE_OPTION	Char	1		Upper	Active Opt	Pos Active,Inactive,Both Optn
POS_ACTIVE	Char	1		Upper	PosnActive	Position Active
POS_INACTIVE	Char	1		Upper	PosnInact	Position Inactive
POS_REPORT_LEVEL	Nbr	2			Rpt Lvl	Pos Hierarchy Report Level
POS_VACANT_REQUEST	Char	1		Upper	Vacant Rqt	Request for Vacant Pos Report
POS_EXCEPT_OVERRIDE	Char	1		Upper	Exe/Override	Position Exception Override
JOB_REQUISITION#	Char	6		Num	Job Req #	Job Requisition #
RQMTS_SRCH#	Char	6		Num	Rqmt Srch#	Requirements Search #
JOB_CODE	Char	6		Upper	Job Code	Job Code
DEPTID	Char	10		Upper	DeptID	Department
REPORT_CHOICE	Char	1		Upper	Rpt Choice	Output Report Choice
POPULATION	Char	1		Upper	SrchSkill	Search Population-Skill Match
EE_REPORT_YEAR	Nbr	4			EE Year	Empl Equity Reporting Year
BUDGET_START_DT	Date	10			Start Dt	Budget Period Start Date
SAL_ADMIN_PLAN	Char	3		Upper	Sal Plan	Salary Administration Plan
INCR_START_DT	Date	10			Start Dt	Start Date for Step Increments
INCR_END_DT	Date	10			End Dt	End Date for Step Increments
JOB_EFFECTIVE_DT	Date	10			Job EffDt	Job Effective Date
COMPANY	Char	3		Upper	Co	Company
PAYGROUP	Char	3		Upper	Group	Pay Group
PERCENTCHG	Nbr	5.2			Change%	Percent of Salary Change
AMOUNTCHG	Nbr	5.2			Amt Chng	Amount of Salary Change
EMPLID	Char	11		Upper	ID	EmplID

Figure 27.3 The RUN_CNTL_HR Run Control record

As you can see from the figure 27.3, this record contains not just the fields necessary for our report. It is also used as a placeholder for most HRMS report input parameters. This does not mean, of course, that all record fields have to be used in every single report. If you know that your report input parameters are among the fields in this record, you can safely use the record as your report application Run Control record.

Let's see if we can use the RUN_CNTL_HR record for our Problem Tracking Status report program. Our program accepts two parameters: As Of Date and Problem Status. The first one, As Of Date, is present in the RUN_CNTL_HR record. The second parameter is our custom field, therefore, we won't find it in the delivered record. We can possibly use any other character type field as a placeholder for our Problem Status field, but if we want to do specific field edits to verify the proper entries, we would be better off creating our own custom Run Control record. Later on, we may want to add other fields to this record and use the record for other custom reports.

The safest way to create our own custom Run Control record is, of course, to clone an existing one. Since the RUN_CNTL_HR record is too big, and, if cloned, will require some effort in deleting all unused fields, let's rather use another record as a template, the PRCSRUNCNTL. This is a PeopleSoft-delivered record used for reports with no application-specific input parameters (figure 27.4).



Field Name	Type	Len	Format	H	Short Name	Long Name
OPRID	Char	8	Mixed		Operator	Operator Id
RUN_CNTL_ID	Char	30	Mixed		Run Cntl	Run Control ID
LANGUAGE_CD	Char	3	Upper		Lang Cd	Language Code
LANGUAGE_OPTION	Char	1	Upper		Lang Option	Language Option

Figure 27.4
The PRCSRUNCNTL record is used for reports with no application specific input parameters

The fields OPRID and RUN_CNTL_ID are the PRCSRUNCNTL record key fields. The LANGUAGE_CD and LANGUAGE_OPTION fields are used in global development projects. The default value of the LANGUAGE_CD field depends on your operator ID. The LANGUAGE_OPTION tells the system if you are allowed to change the LANGUAGE_CD field.

After saving this record as MY_RUN_CNTL, deleting the LANGUAGE_CD and LANGUAGE_OPTION fields, and adding the fields that we need as our report input parameters, the record will look like that in figure 27.5.

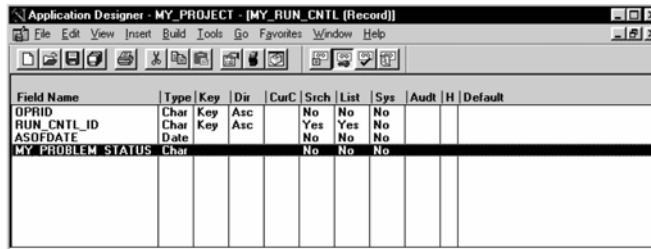


Figure 27.5
Creating a custom Run
Control record:
MY_RUN_CNTL

Since we've created our own record, we should not forget to add a description to identify the customizations.

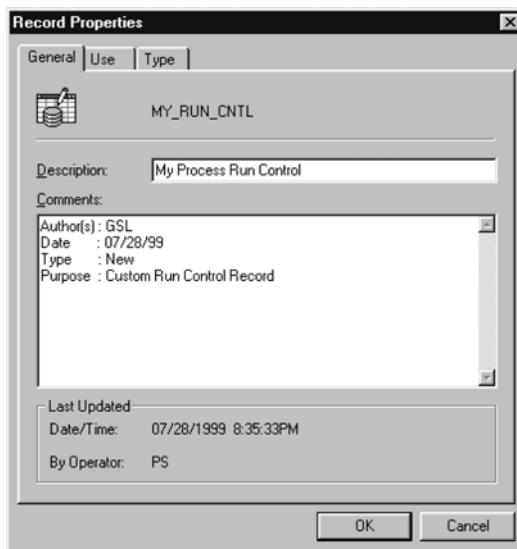


Figure 27.6
Entering record properties

After saving the record again, let's execute the Build option to create the database level table (figure 27.7).

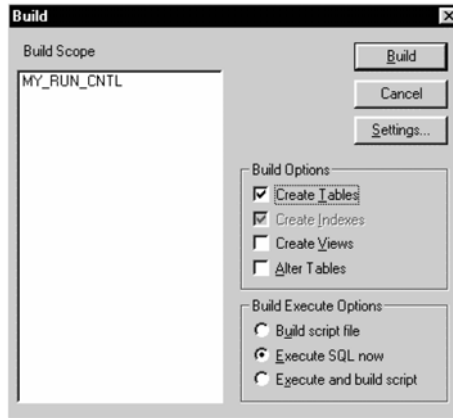


Figure 27.7
Building our custom Run Control record

As soon as the record is saved, it is automatically added to our project.

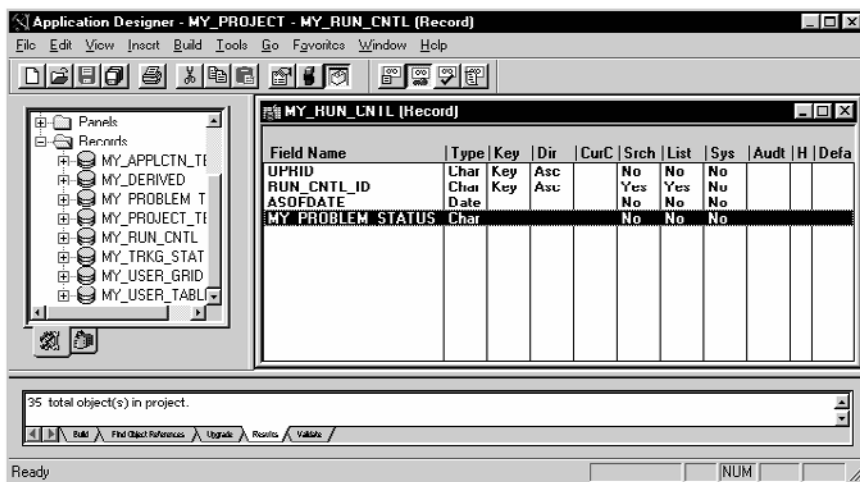


Figure 27.8 MY_RUN_CNTL Run Control record is added to a project

Now we can save the project and go to the next step of creating a Run Control panel.

27.2 CREATING A RUN CONTROL PANEL

PeopleSoft delivers many Run Control panels along with its applications. Therefore, if your report is using the same parameters as one of the PeopleSoft-delivered reports,

it makes perfect sense to re-use the delivered panel. Since our report uses custom fields as input parameters, we create a new Run Control panel for it.

As we discussed in parts 2 and 4 of this book, each panel should have at least one record linked to it. In our particular case, we already know what record we are supposed to link to the Run Control panel since we just created it: it's our MY_RUN_CNTL record, which will hold all input parameters for our report.

Just as we did for the record creation, let's clone the Run Control panel that does not accept any parameters, then add our two fields to it. The panel name is PRCSRUNCNTL.

Let's check the panel structure in the Application Designer.

Navigation: Go →PeopleTools →Application Designer →File →Open.

Enter Panel as an object type, and PRCSRUNCNTL as a panel name. Press ENTER. You will see the panel that appears in figure 27.9.

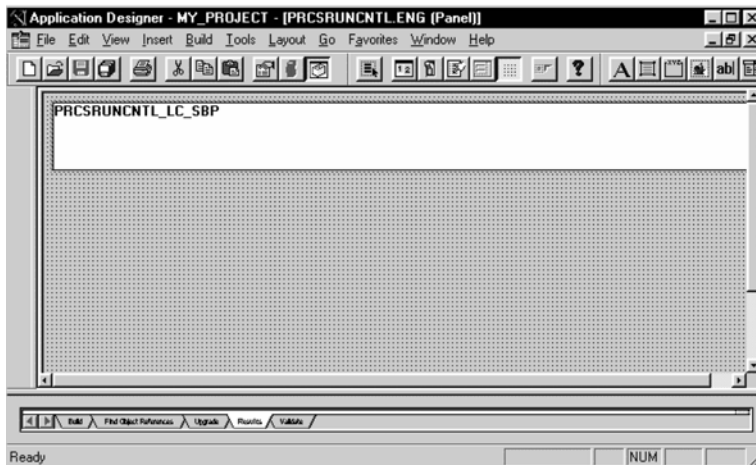


Figure 27.9 The standard Run Control panel with no input parameters


The panel contains a subpanel named PRCSRUNCNTL_LC_SBP. All the PRCSRUNCNTL panel fields are located inside of this subpanel. If you select Layout, Test Mode, or click on the Test Mode button , you can see all the panel fields (figure 27.10).



Figure 27.10 The PRCSRUNCNTL panel in test mode

Of the three fields in the panel, operator ID and the Run Control ID are the key fields automatically populated from the operator ID and the Run Control ID you entered. The third field, Language, is optional. If the operator does not select any Language from a prompt, it defaults to the operator's default language.

Since we are using this panel as a basis for cloning, we first save it as MY_RUN_CNTL_PRB01 panel and then modify it (figure 27.11).

After pressing the OK button, we add our custom fields to the panel under construction. This time we use our project to speed up the development. Double-click on the MY_RUN_CNTL record from the project workspace window. You can see all the fields in our record. Click on the As Of Date field and drag it to the panel space. Drop the field on the panel where you want the field to be placed. Repeat the same procedure for the MY_PROBLEM_STATUS field. Figure 27.12 shows our new panel.

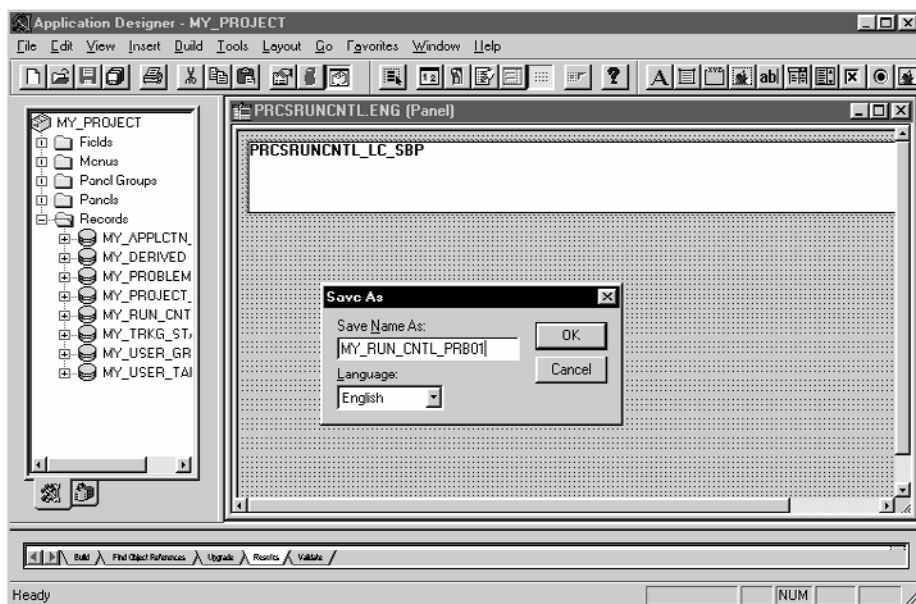


Figure 27.11 Cloning the PRCSRUNCNTL panel

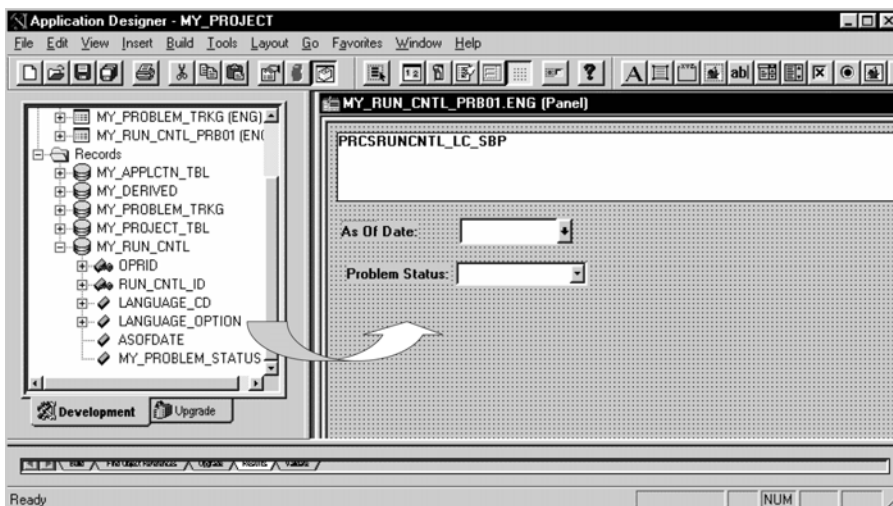


Figure 27.12 Dragging fields from the project to the panel

Since we modified our panel, let's not forget to save it.
Let's check the panel's layout. Select Layout → Order (figure 27.13).

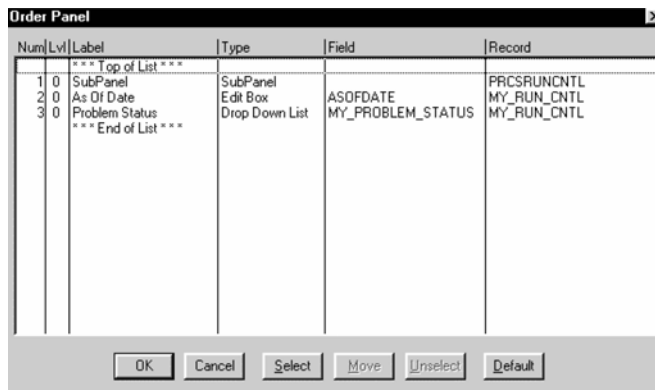


Figure 27.13
Verifying the panel's layout

Our simple Run Control panel has only Level 0 fields. The standard subpanel is linked to the PRCSRUNCNTL record that has the same keys as our MY_RUN_CNTL record. Our two fields are in the correct order and belong to our custom record. The last thing we need to do is to update the Panel's properties.

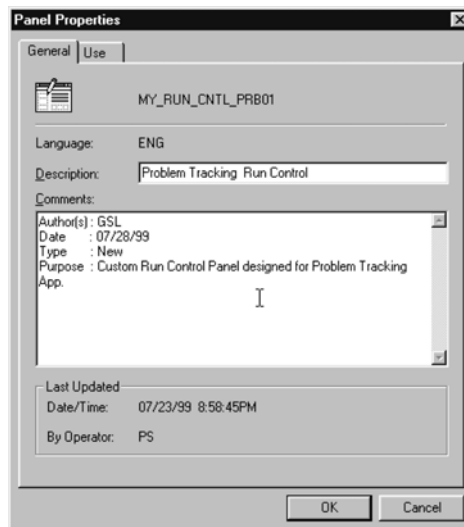


Figure 27.14
Specifying panel's properties

Our panel design is complete. In order to place it in a menu, we need to create a panel group.

27.3 CREATING A PANEL GROUP

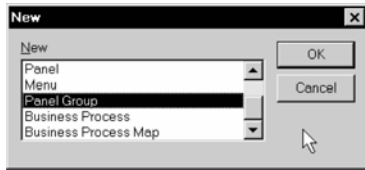


Figure 27.15 Creating a new Panel Group

After a panel is selected or created, it must be added to a panel group before you can attach it to a menu. A panel group is actually a link between the panel and the menu. Multiple panels may exist within a single panel group.

Let's create a new panel group by selecting File, New (in the Application Designer menu), and double-clicking on Panel Group in the New dialog (figure 27.15).

The Application Designer Panel Group screen appears (figure 27.16).

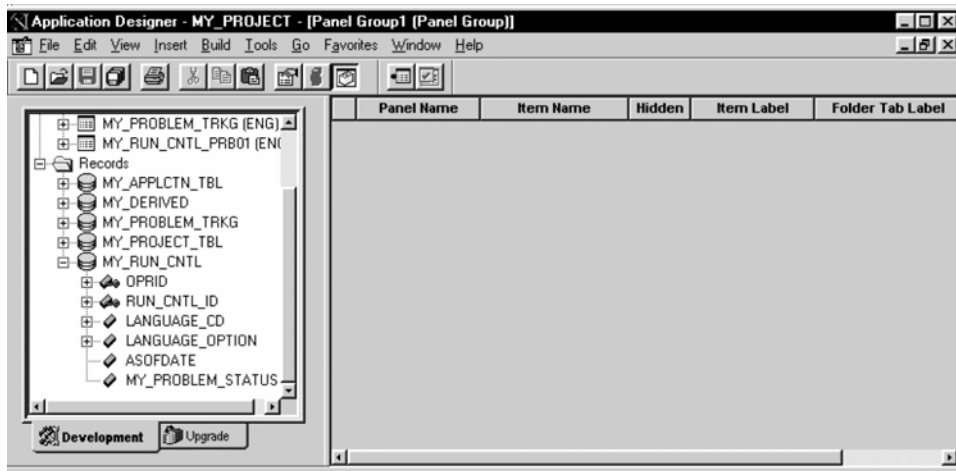



Figure 27.16 The Panel Group panel

We now need to add our panel (MY_RUN_CNTL_PROB01) to the panel group. To do this, you can either click on the Insert Panel button  on the toolbar, or select Insert, Panel into Group, or use the drag-and-drop technique from the project window.

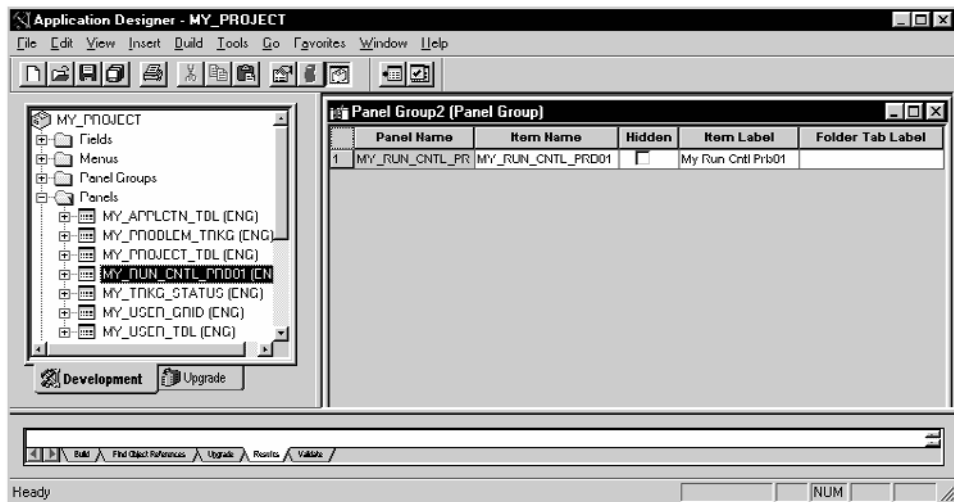


Figure 27.17 Dragging a panel from a project workspace to a panel group

Each panel in a group has a set of properties. The MY_RUN_CNTL_PR001 panel has been added to the panel group with its properties set to their default values. Let's change these values to make them more meaningful.

The Item Name is used for informational purposes only, but it must be unique within the panel group. We'll specify our own name as Problem_Tracking.

The next property column is Hidden. You only check this value On if you need the panel to be hidden from the user's view. We'll leave the value of this column Off. You can have several panels in a panel group with the Hidden value set to Off and one or more panels with the value set to On. This technique is used when you need to bring to the buffer certain fields from some panels, but you don't want to display these panels to users.

The Item Label column is your panel name as it will appear on the menu. It will also be displayed at the bottom of the panel and as the default Folder tab label. Right now, it is named RUN CNTL PRB01, which is not very meaningful. Let's call it Problem Status Report.

The Folder Tab Label is used to identify the Folder tab when the Panel Group is selected. Let's name it Problem Status for our task.

After we enter all the values, our panel group definition looks like that shown in figure 27.18.

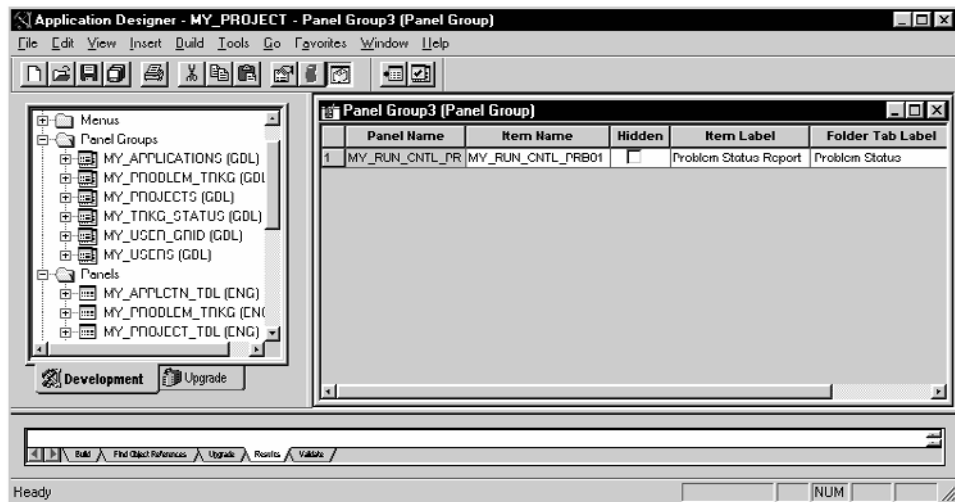


Figure 27.18 Setting panel group properties

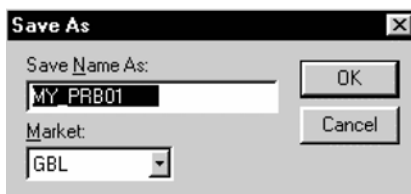


Figure 27.19 Saving a panel group

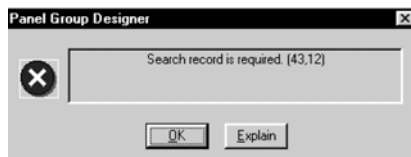


Figure 27.20 Panel group error window

Now, let's try to save our panel group. After clicking on the OK button, an error message pops up (figure 27.20).

The Panel Group designer reminds us that we cannot save our panel group yet. We have not completed the design. We need to set the properties for the entire panel group, including search records, update and data entry actions, and detail panel information.

As we discussed in part 4, in order to allow our users access to the panels, the Search record has to be attached to the panel group. The search record that you select should contain all the keys that your user needs in order to retrieve rows displayed on the panel. Based on the actions that the user selects (Add,

Update/Display), the Application Processor creates a prompt dialog box, which contains all the key fields in the search record. In our case, since we are defining a search record for a Run Control panel, the key fields are the operator ID and the Run Control ID. When the user selects an action to run a report, he/she is usually presented with two options: Add and Update/Display. When the Update/Display option is selected, the report is run under an existing Run Control ID. If the Add option is selected, a new Run Control ID is created to be used with the report. And, since the dialog box for operator should contain only the Run Control ID, you can

always specify the standard PRCSRUNCNTL record as a search record for your report's panel group, no matter what application-specific parameters are defined for your report. This greatly simplifies the task of creating a panel group for reports.

PeopleSoft-delivered reports use the same search record: PRCSRUNCNTL.

Let's get back now to figure 27.20. After clicking on the OK button, the Panel Group Properties window appears. Let's fill in the Use tab as well as the General tab with the panel group description.

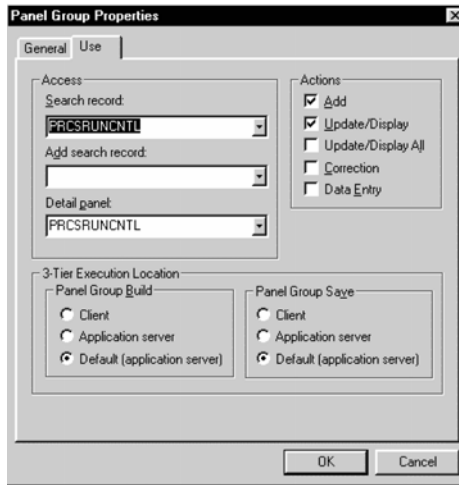


Figure 27.21
The Panel Group Properties

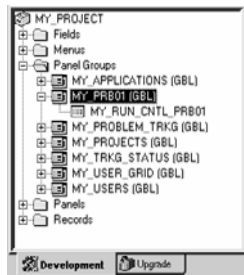


Figure 27.22 Saving
the MY_PROB01 panel
group automatically
adds it to our project

After all information is entered in the two tabs of Panel Group Properties, we can save our panel group. Select File, Save, and enter the new panel group name as MY_PROB01. Our panel group is automatically added to the project (figure 27.22).

27.4 SELECTING A MENU FOR YOUR REPORT

The next decision you must make concerns the menu under which you will run your report. During the course of developing our Problem Tracking application, we

already created a separate menu item named Problem Tracking. Let's open this menu item and add another item, Reports, to the menu bar. In order to do so, just click on the empty rectangle (figure 27.23), and specify the new Bar Item properties.

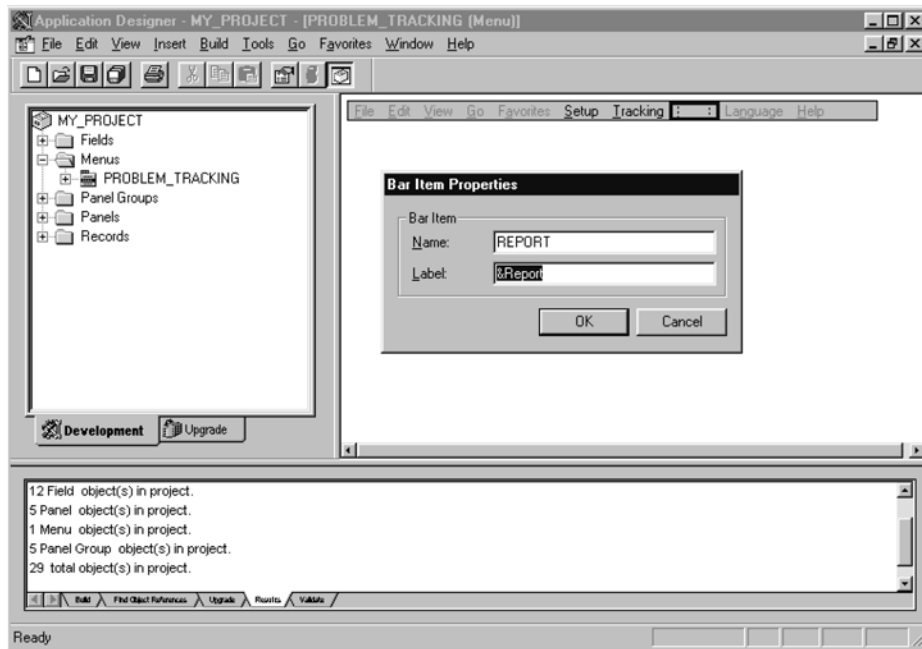


Figure 27.23 Adding a new bar item to the menu

Next we add a new menu item named Status Report to the Problem Tracking menu under the Report menu bar.

To create a new menu item, double-click on the empty rectangle on the menu. The Menu Item Properties dialog appears (figure 27.24).

On the panel shown in figure 27.24, click on Select and add our MY_PROB01 panel group to the Status Report menu item.

After the new menu item is created, you have to decide who will be able to access it. Only users who belong to the proper operator class should be granted access to the new menu item.

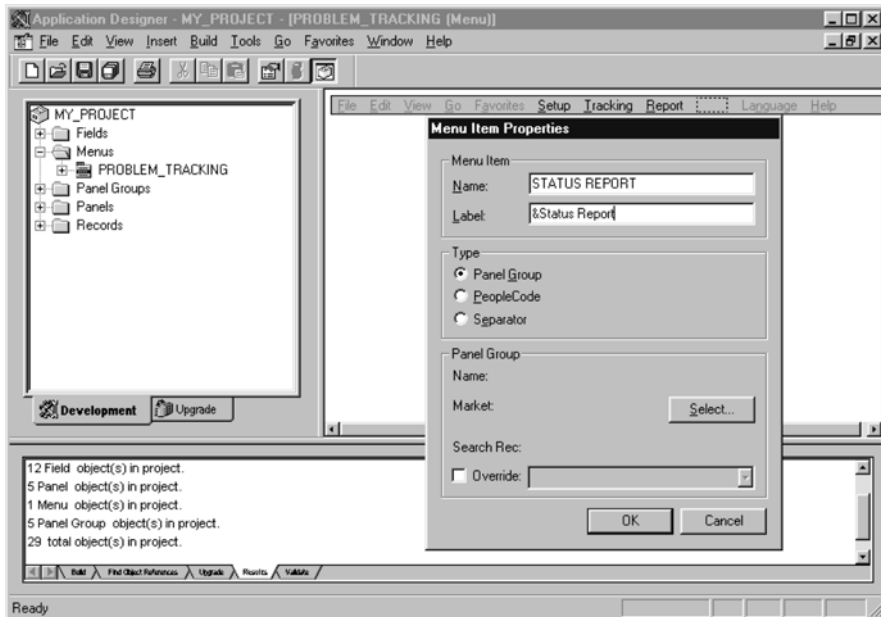



Figure 27.24 Adding a new menu item to the Report menu bar

27.5 GRANTING SECURITY ACCESS

Since we created a new menu item, we have to allow certain users to access it.

First, you need to grant access to the ALLPANLS operator class, which will be used in testing your application. Select File, Open and enter ALLPANLS as shown on the screen in figure 27.25.

Press OK and click on the Menu Items icon  to display the list of all available menus for this operator class (figure 27.26).

Navigation: Go →PeopleTools →Security Administrator.

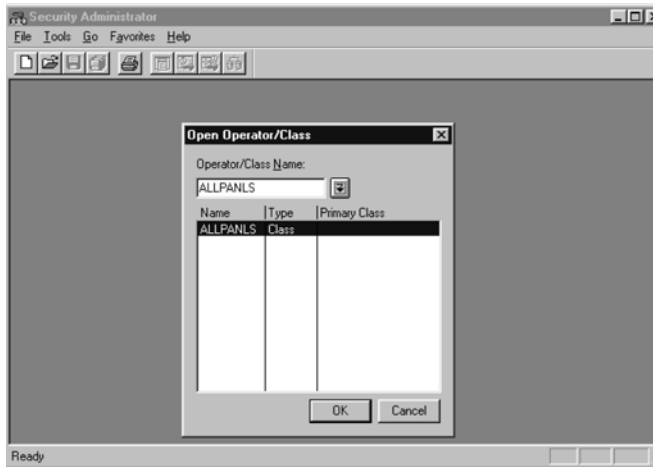


Figure 27.25
Opening the security panel for ALLPANLS operator class

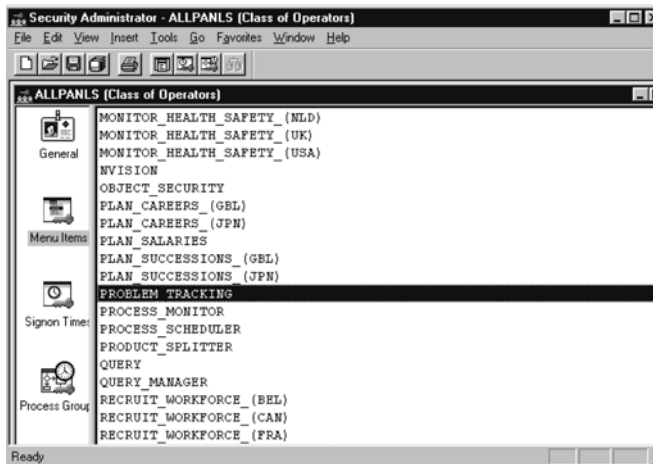


Figure 27.26
Menu Items to which ALLPANLS has access

We placed our new menu item under the Problem Tracking menu. In order to see all menu items under this menu, let's double-click on Problem Tracking. The system returns a list of all available menu items under the Problem Tracking menu (figure 27.27).

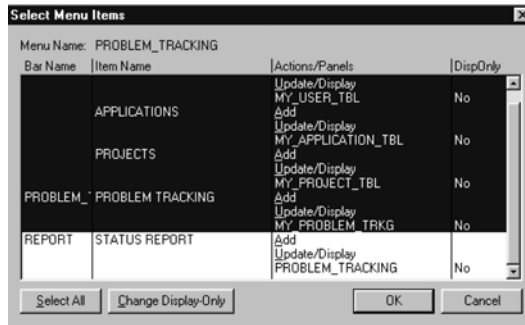


Figure 27.27
Selecting the Problem Tracking menu from all menus available to the ALLPANLS class

The newly created menu bar Report and the item Status Report are not highlighted and, therefore, not available to any operator from the ALLPANLS operator class.

Let's highlight all three lines that belong to the Status Report item by clicking on each line. Press OK to make our new menu item available for the ALLPANLS operator class.

27.6 TESTING YOUR CHANGES

So far in this chapter we created the following new objects:

- Run Control record, MY_RUN_CNTL
- Run Control panel, MY_RUN_CNTL_PRB01
- panel group, MY_PRB01
- menu bar, Report
- menu item, Status Report

Remember, our goal was to allow users to execute our SQR report from the online panels via PeopleSoft Process Scheduler. Is this all we need to do? If you recall all the steps we went through in this chapter, you'll note that our SQR report was never linked to any of our new objects. We prepared a Run Control record to hold the input parameters; we created a panel to accept these parameters online; and we even created a menu from which to display the panel. The last step is to create a process definition in order to attach our SQR to the panel group. We will discuss all the steps of creating a process definition in the following subchapter. Before we do this, we can test all the objects we developed without executing our SQR. It is important to make certain that our online components work properly.

Figure 27.28 shows the new menu item.

Our new menu bar and menu item look as we planned. Let's select the Add action and make sure that our new panel group and panel are working as well. After entering a new Run Control ID, MY_STATUS_01, the system displays the panel shown in figure 27.29.

Navigation: GO →Problem Tracking →Report →Status Report



Figure 27.28 The newly added menu bar Report and menu item Status Report

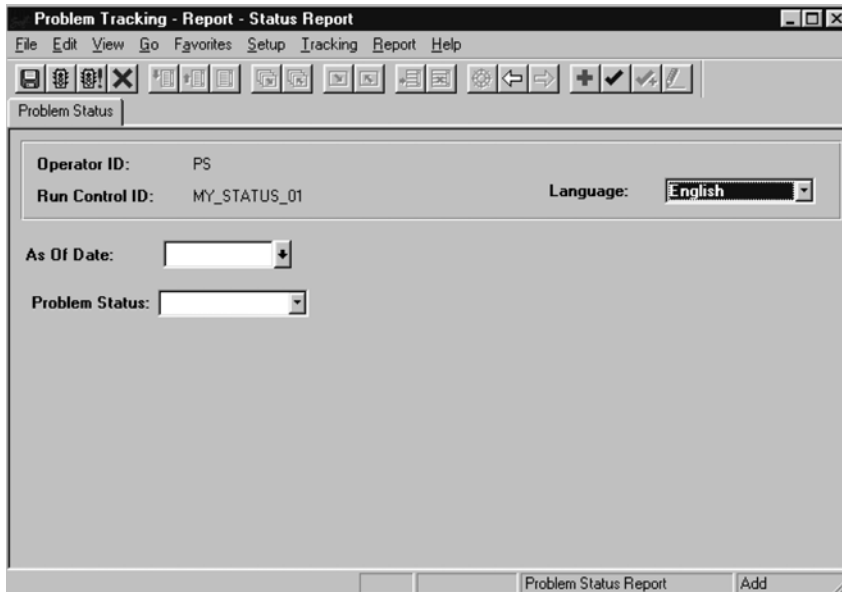


Figure 27.29 Invoking the Status Report panel

If you click on the As Of Date field, the date selection calendar should pop up, since this field was defined as a date. Let's test this field first.

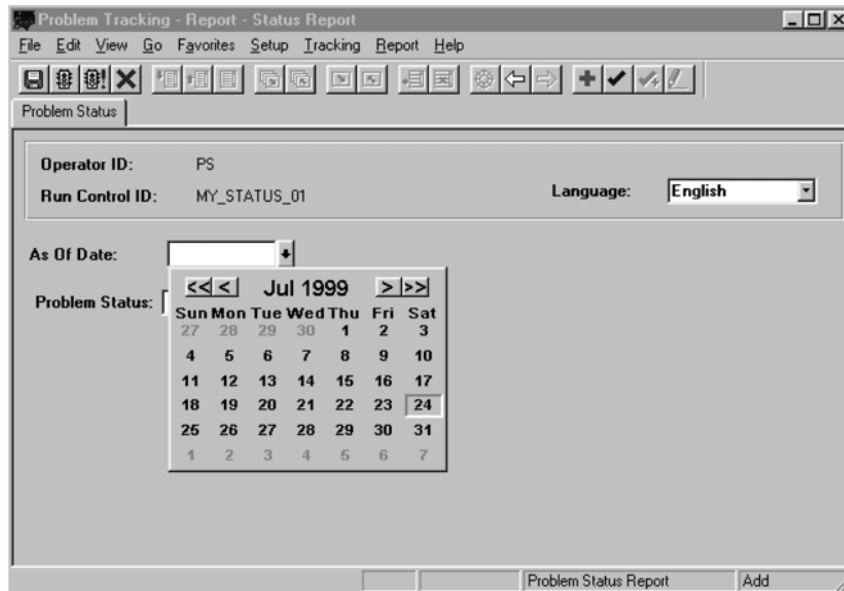


Figure 27.30 Selecting the date in As Of Date field from the pop-up calendar

As you can see, we can select any date from the pop-up panel. Let's click on the 24th of July. After the As Of Date is entered, move on to test our next field, the Problem Status (figure 27.31).

The Problem Status field prompts us to select from any of the values in the edit box. Let's select *Initiated* and save our selections. Where will the values be kept? The system saves all the panel's values in the record attached to this panel. In our case, that record is the MY_RUN_CNTL record that we created for this purpose. You can use your database-specific native SQL tools to select data from this table in order to be sure that our panel is working properly.

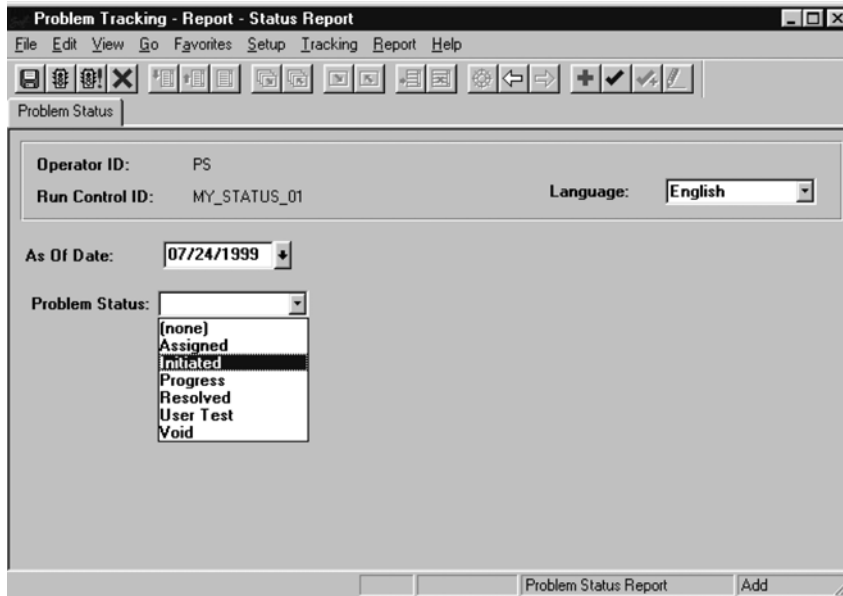


Figure 27.31 Selecting the Problem Status value for our report

Figure 27.32 shows the selected row from our Run Control table PS_MY_RUN_CNTL. (Remember, you have to add the prefix ‘PS’ to the tables when accessing them via your database native SQL tools.) All the fields we entered are properly saved. It is a good idea to always verify your Run Control tables when creating new panels to make sure they are populated correctly.

And now we are ready to create a process definition for our SQR report.

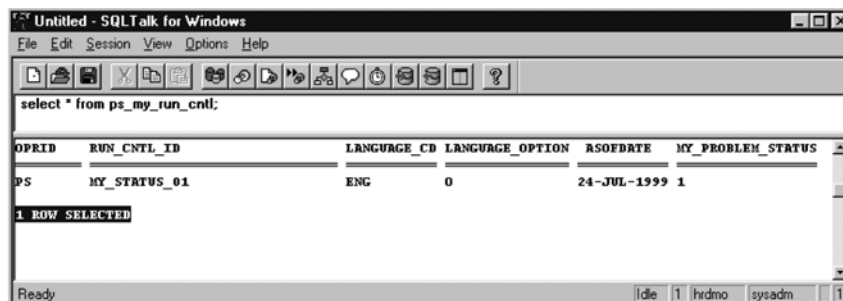


Figure 27.32 Selecting the Run Control information

27.7 CREATING A PROCESS DEFINITION FOR THE PROBLEM STATUS REPORT

Every process run under the PeopleSoft Process Scheduler needs a process definition to specify the process attributes and link the process to the appropriate panel group. We will go through all the steps of creating a process definition for our Problem Status report.

The system displays the Process Definition dialog box (figure 27.33).

Navigation: Go →PeopleTools →Process Scheduler →Use →Process Definitions →Process Definitions →Add



Figure 27.33 Assigning a type to your process

As you can see from the dialog box, we have to select the appropriate process type for our process. Please note that the valid type for our process is SQR Report, not SQR Process.

NOTE If you select SQR Process instead of SQR Report for the Process Type, the Process Scheduler will not pass the operator ID and Run Control ID to your program, and the program will not work correctly, unless you specify operator ID and Run Control ID as additional parameters.

The process name must be the same as your program name: MYPROB01. Please note that no SQR extension is needed (figure 27.34).



Figure 27.34
No extension is needed when entering the program name

After you press OK, the system displays the Process Scheduler Process Definitions panel group (figure 27.35). This panel group consists of the following three panels:

- the Process Definitions panel
- the Process Definitions Options panel
- the Panel Transfers panel.

The Process Definitions panel tab is the only one you have to fill in; the other two tabs in this panel group are optional.

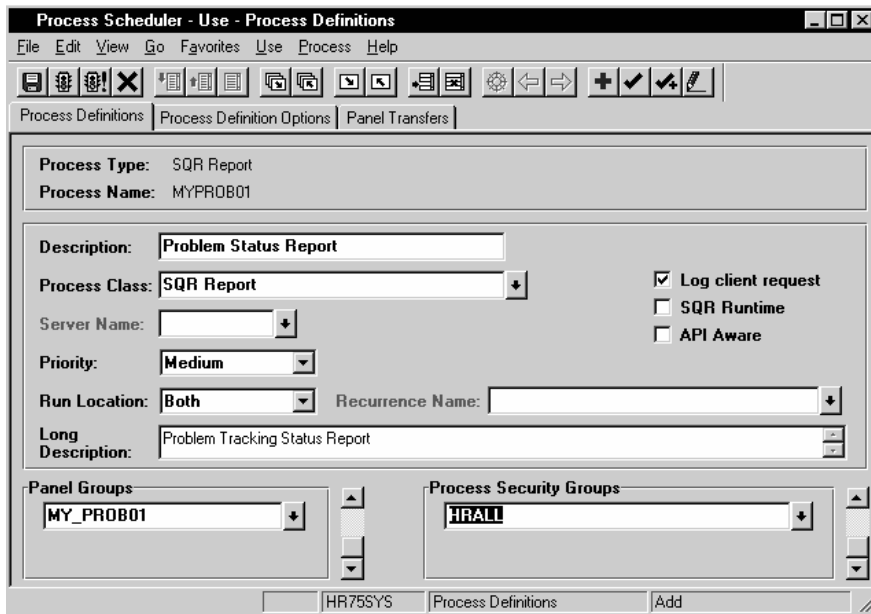


Figure 27.35 The Process Definitions panel

27.7.1 The Process Definitions panel

In the Process Definitions panel, you have to enter information about your process. Let's look at all the fields on the panel shown in figure 27.35.

- The *Description* will be displayed along with your process name on the Process Request panel, so make it meaningful.
- The *Process Class* must be a valid process class from the selection list. In our case, it is SQR Report.
- The *Server Name* (Optional) is specified if you plan to always run your process on a particular server. Otherwise, leave it blank. If, for example, you have both the Unix Server and the NT Server available to run your process, and you do not specify the server name on this panel, users will be able to select the server of their choice on the Process Request panel. If a user leaves the server name blank on the Process Request panel, the system will automatically find the first available server that can process the request for this process class.

TIP The server name can be specified only if the run location is *Server*.

- The *Priority* can be set to *Low*, *Medium*, or *High*. If several processes are queued on a particular server, the system will be using this selection to decide which process should be initiated first. This parameter is applicable for processes that run on a server only.
- The *Run Location* (Optional) can be *Server*, *Client*, or *Both*. If either *Server* or *Client* is selected, it specifies the run location for your process request. If set to *Both*, the process is initiated on the Run Location set in the Process Scheduler Request panel. Note that this selection takes precedence over the Process Scheduler Request specification. This means that, if you select *Server* here, the process will be scheduled to run on the server only, regardless of what the user specifies in the Process Scheduler Request dialog box.
- The *Recurrence Name* (Optional) can be selected only for processes that run on a server. The recurrence definitions are created in the Process Request dialog. All previously created recurrence definitions are shown in the drop-down list for the Recurrence Name field. Note, if you specify the Recurrence Name here, this does not mean that the process will automatically start and run according to the specified recurrence definition.

TIP In order to schedule your process for recurrent execution it has to be started manually from the Process Request Dialog Panel for the first time.

(Please see more about using run recurrences for your process in chapter 30.)

- The *Long Description* (Optional) is used for your process description.

- An *API Aware* process is a process that updates the Process Request table (PSPRCSRQST) with the process run status (Error, Success, and such), completion code, message set, and message number. This allows the system to perform a Commit or a Rollback, depending on the run status. Based on the process execution results, the system displays a standard or custom message on the Process Monitor's Process Request Detail panel. Not every program is API Aware. You have to add certain logic to your SQR program to make it API Aware.

WARNING Turning the API Aware flag On does not automatically make the process API Aware.

We'll discuss the process of making an SQR program API Aware in detail in the next chapter, but please note that, if your program is not API Aware, the flag must be turned Off.

As you can see, we turned this flag off for MYPROB01 process definition, since we did not place any special code to make our program API Aware—yet. (We'll do this in the next chapter.)

- If *Log Client Request* is on, the system logs the request on the Process Request table every time the process is run on the client. This is useful as an audit trail. Note that, for all server run requests, logging is always performed. By default, it is turned On for all API Aware processes.
- The *SQR Runtime* is checked when you want the system to append the .sqt extension to the process name (used for precompiled SQR programs). It will use the SQT working directory. For our Problem Status program, this option should be turned off.
- The *Panel Group* is used to specify the panel group from which you want to run your process.

In our case, the panel group is MY_PROB01 because we created this panel group to run the Status Report. Note that, in order to link your process to a panel group, this panel group must be created prior to creating the process definition.

TIP Make sure you enter the correct panel group name. PeopleSoft does not edit this field. If you misspell it, users will not be able to run your process.

To avoid the problem, click on the panel group and press CNTL + F4. The system will display a list of all available panel groups.

Optionally, you can specify more than one panel group for your process by inserting additional rows in the Panel Groups box. In this case, the process will appear on all selected panel groups.

- The *Process Security Groups* define operator classes or operators that have permission to submit this process. At least one process security group must be specified. You can allow multiple process security groups to run your process. You have to specify the process security groups that belong to your user's operator class. If you specify a security group here, but do not give permissions to some operators to use this group, the process will not be visible to those operators.

Let's make certain that the HRALL process security group belongs to the ALLPANLS class. Switch to the Security Administrator panel, click on the Process Groups icon within the panel, and check if the operators who belong to the ALLPANLS class are allowed to use the HRALL process group. HRALL must be among other process security groups under the ALLPANLS class (figure 27.36).

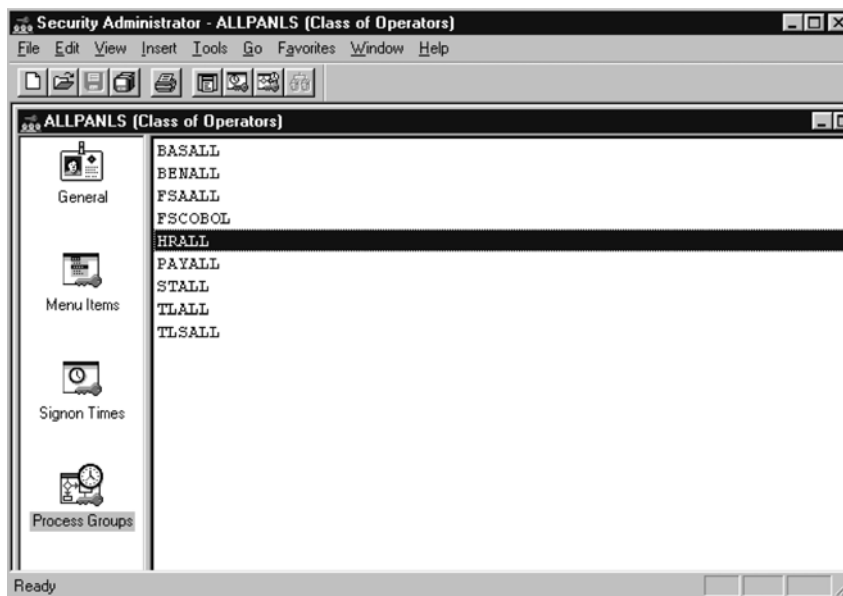


Figure 27.36 The authorized Process Groups for the ALLPANLS operator class

27.7.2 Process Definition Options panel

The second panel of the Process Scheduler panel group, the Process Definition Options panel, (figure 27.37) is optional.

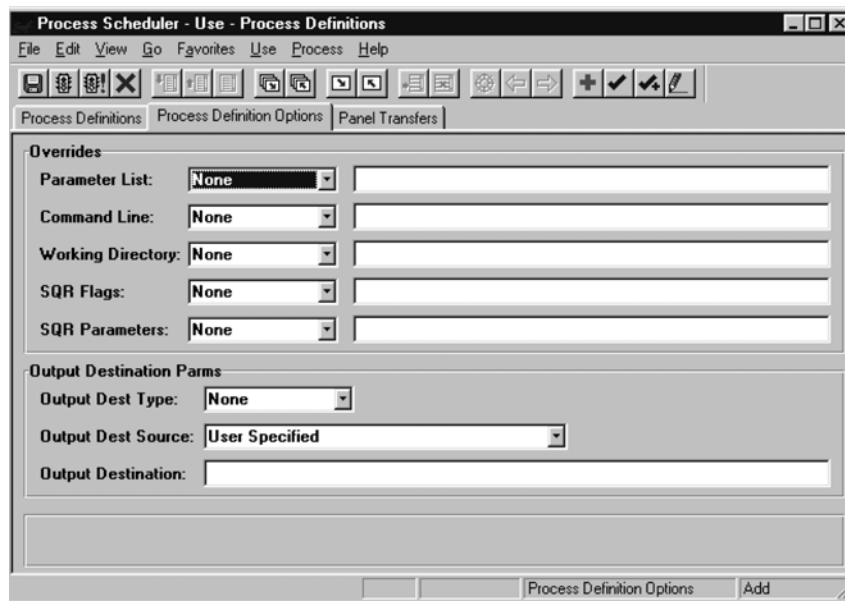


Figure 27.37 The Process Definitions Options panel for our process definition

This panel is used to modify the process parameter list, command line, working directory, and SQR flags and parameters. It is also used to change the Output Destination parameters.

The drop-down lists for each parameter allow you to preface, append, or override each parameter for your process. Suppose you want to invoke the SPF Viewer after generating your program.spf file. All you need to do is append the `-ZIV` flag to your SQR Flags parameter in your Process Definition Options panel (figure 27.38).

To illustrate another useful example (figure 27.38), we appended two parameters to the standard parameter list: `MY_DERIVED.MY_USER_ID` and `MY_USER_TABLE.NAME`. This is a simple and efficient technique that allows you to pass the parameters directly from your panel to the SQR program. The parameters are coded in the form of `Record.Field`. Please note that the SQR program must issue two additional input commands in this case to accept these two parameters.

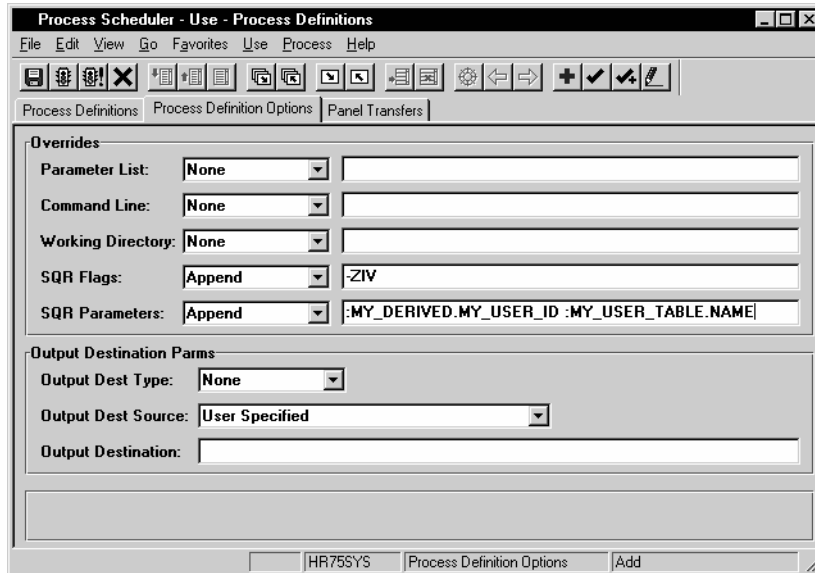


Figure 27.38 An example of the Process Definitions Options panel with additional parameters

TIP When appending additional SQR parameters via the Process Definitions Options panel, your SQR program should contain the Input commands to accept these additional parameters.

TIP For SQR programs, Output Dest Source must be set to User Specified.

For our process we won't be using any of the panel fields.

27.7.3 Panel Transfers panel

The Panel Transfers panel is a part of PeopleSoft Workflow. Also optional, it allows you to transfer to the specified panel from the Process Monitor after your process is successfully completed. You can specify directions of transfer and menu actions in this panel.

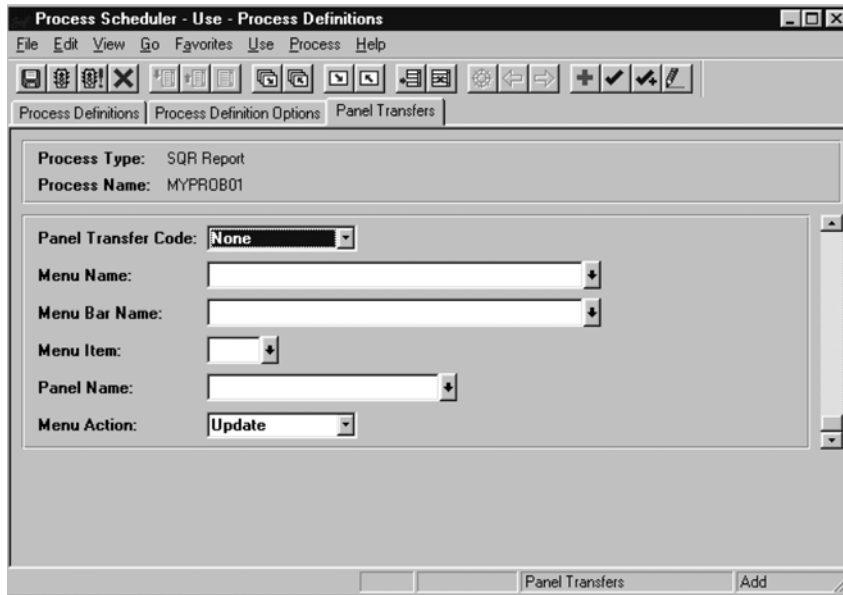


Figure 27.39 The Panel Transfers panel

For our sample program, we will leave this panel unchanged.

27.8 SPECIFYING THE PROGRAM DIRECTORY

Your last task is to place your program into the right directory so that the Process Scheduler will be able to find it. How do we know where the Process Scheduler expects to find the program? Let's take a look at the PeopleSoft System Configuration Manager.

Navigation: Edit → Preferences → Configuration → Process Scheduler

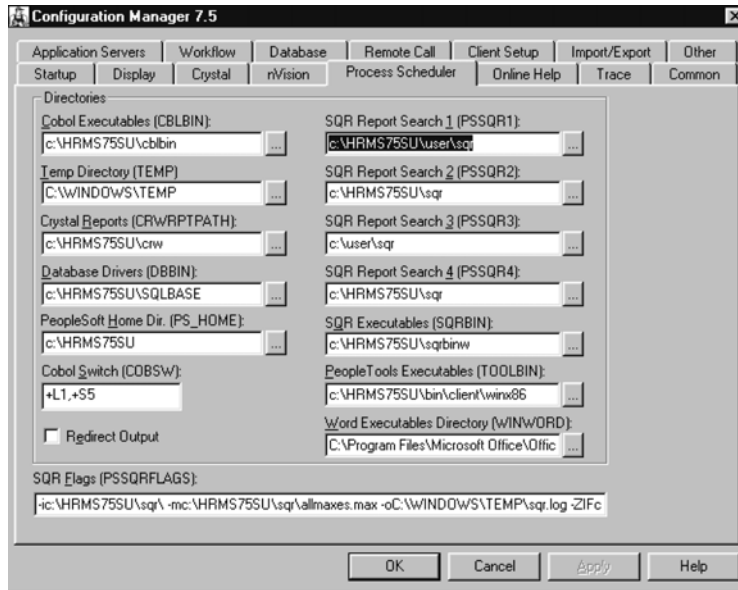


Figure 27.40 The PeopleSoft System Configuration panel

You have to ensure that your SQR program is in the path specified by either PSSQR1, PSSQR2, PSSQR3, or PSSQR4 search path variables. Let's copy the MYPROB01.sqr program into c:\hrms75su\user\sqr.

27.9 TESTING YOUR PROCESS DEFINITION

Now, we are ready to run our SQR program.

Select the Status Report from the Problem Tracking Menu. The system displays a Run Control prompt as shown in figure 27.41.

Note that this time we select the Update/Display option since we already created our Run Control record and therefore can reuse it. Let's select the MY_STATUS_01 Run Control ID.

The system displays the Run Control panel shown in figure 27.42.

Navigation: Go → Problem Tracking → Report → Status Report → Update/Display

Run Cntl	Lang Cd
1 MY_RUN01	English
MY_STATUS_01	English

Figure 27.41
Run Control prompt

Figure 27.42 The Problem Status Report Run Control panel

As you can see from figure 27.42, all the parameters in this panel are already set up. This is because the information is retrieved from our Run Control record that is attached to this panel. We are ready to execute our program for the first time from the online panel. Click on the Traffic Light, and you will be presented with the process request panel (figure 27.43).

Description	Name	Process Type Descr
Problem Status Report	MYPROB01	SQR Report

Figure 27.43
The Process Scheduler Request panel for the Problem Status report

On the lower portion of the Process Scheduler Request panel, you can see that MYPROB01 is displayed as a program name. This means that the process definition that we created earlier correctly attached our SQR program to the panel group.

Figure 27.44 SQR Prompts for the As Of Date value

Let's click OK and start testing our program. The program should run to the end without any problems.

Our program displays the first input prompt (figure 27.44).

After entering a valid date, we see another prompt (figure 27.44) for the problem status value.

Figure 27.45 SQR Prompts for the Problem Status value

Once we enter the problem status value, the program runs to the end. You may be asking yourself a question: "Why do we need to enter the same input information in both the Run Control panel and the prompt boxes?" The reason for this strange behavior is that our program does not know that it runs under the Process Scheduler. If the program is called for execution from the Process Scheduler, it

should accept the input parameters from the online Run Control panels; otherwise, it should prompt the user to enter the input parameters via the Input command. In our next chapter, we will discuss this in detail, and will modify our SQR program to work correctly no matter under what environment it runs.

Let's verify the process execution status on the Process Monitor panel (figure 27.46).

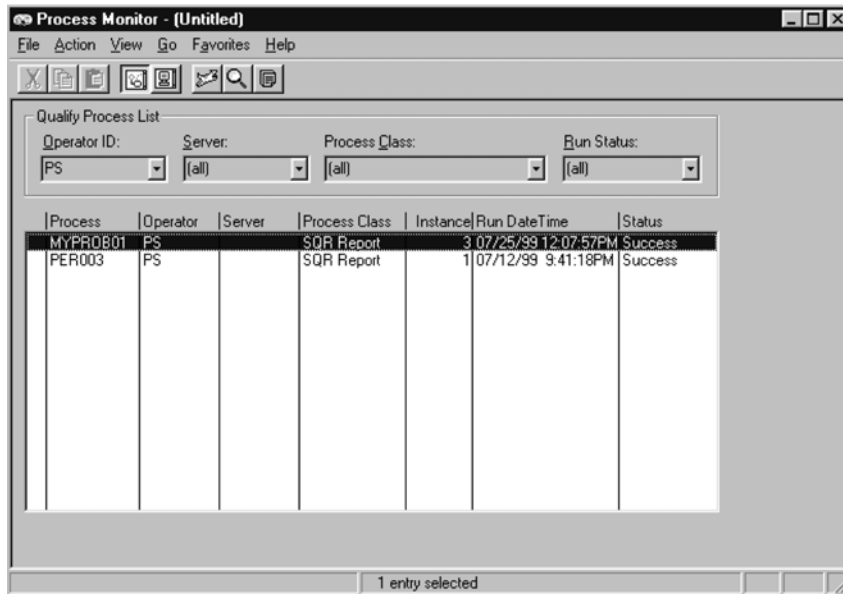


Figure 27.46 The Process Monitor panel

The status of the MYPROB01 process on the Process Monitor panel shows Success. This sounds good, but it does not mean that your project is finished. Even if our program fails, the Process Monitor has no idea of the program execution status. Remember that we just took an SQR program developed with no PeopleSoft interface code and plugged it into the PeopleSoft Process Scheduler. This allowed us to initiate and run the program from the PeopleSoft panel. The Status Report output has been created, but the Process Monitor's process status has not been updated. Therefore, PeopleSoft has no idea about the return code of the process. In the next chapter, you will learn how to solve the problem by making your program API Aware.

KEY POINTS

- 1** Any Run Control record must have the operator ID and the Run Control ID as its key fields.
- 2** A Run Control record may contain additional fields not used in your program.
- 3** A Run Control panel should be made specific to your application and should contain (or display) only the necessary fields.
- 4** A process definition must be created to link to the panel(s) from which it will be run.
- 5** You can add your SQR Program to an existing menu item or create a new one.
- 6** The appropriate security access must be granted to all operator classes that will be allowed to see the new menu item.
- 7** In order to inform the Process Monitor about the status of your program, you have to modify the program to make it API Aware.



CHAPTER 28

Communicating with the Process Scheduler

- | | |
|---|--|
| 28.1 Using PeopleSoft-delivered SQC files 635 | 28.4 Exercise 3: Accept the As Of Date and problem status parameters from an on-line panel 643 |
| 28.2 Exercise 2: Make your SQR program API Aware 636 | 28.5 Testing your changes 652 |
| 28.3 Creating a new process definition for an API Aware program 640 | |

In most cases, SQR programs that run under PeopleSoft need certain changes. While any SQR program can be executed under the Process Scheduler, only programs that include special code are capable of communicating their status back to the Process Scheduler. In order to allow the Process Monitor to reflect your program status, you have to make your program API Aware.

28.1 USING PEOPLESOFT-DELIVERED SQC FILES

PeopleSoft provides a number of routines that handle the communication between SQR programs and the Process Scheduler. In order to make your SQR program API Aware, you have to add the PeopleSoft-delivered program files (SQC files) that contain these routines to your program. At a minimum, you need to include the STDAPI.sqc and SETENV.sqc files to your program. The STDAPI.sqc, in turn, uses the nested #Include operators that refer to other important API files (figure 28.1). Let's look at two of these files: PRCSDEF.sqc and PRCSAPI.sqc.

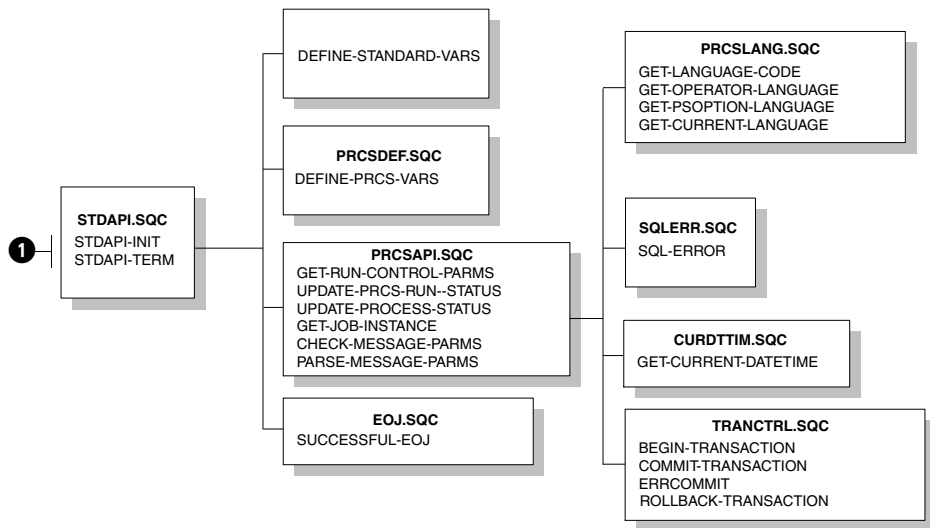


Figure 28.1 The Process Scheduler API SQC files and procedures

- 1 These 2 procedures are called from every API Aware SQR Program.

The PRCSDEF.sqc file includes the `Define-Pracs-Vars` procedure. This procedure initializes all the fields used in API. The PRCSAPI.sqc file includes two important procedures: `Get-Run-Control-Parms` and `Update-Pracs-Run-Status`. The first procedure, `Get-Run-Control-Parms`, retrieves the input parameters (Process Instance, Operator ID, and Run Control ID) and updates the run status of the process request to Processing. The PRCSAPI.sqc, `Update-Pracs-Run-Status` procedure, is designed to update the Process Request table (PSPRCSRQST) upon program completion.

When you run your program from the Process Scheduler, the control parameters that identify your process (the process instance, the operator ID, and the Run Control ID) are passed as a part of the command line. The application-specific input parameters are not passed to the program—these parameters are saved in the Run Control

table. When you run the same program from the SQR dialog box or from the command line, the Get-Run-Control-Parms API procedure does not detect any input values from the Process Scheduler and instead identifies the process as being run from outside the Process Scheduler.

Figure 28.1 lists PeopleSoft-delivered API SQC files and procedures and also shows the location of each API procedure and SQC file.

28.2 **EXERCISE 2: MAKE YOUR SQR PROGRAM API AWARE**

Making an SQR program API Aware involves adding program code to update the Process Request table (PSPRCSRQST) with the program run status (Error, Success, and so on), completion code, error message set, and error message number.

28.2.1 Incorporating SQC files into your program

Let's add the SQC files we just discussed to our Status Report. (The updated program will be called MYPROB02.sqr). To save space, we will show only the modified parts of the program.

```
!MYPROB02.SQR
!Problem Status Report

#include 'setenv.sqc'

!...
! *****
Begin-Program
! *****
do Init-DateTime
do Init-Number
do Init-Report
do Main
do Stdapi-Term

End-Program

! *****
Begin-Procedure Init-Report
! *****
  do Stdapi-Init
  do Ask-Input-Parameters
  do Build-Where
  do Load-Xlats
End-Procedure

!...
! *****

#include 'stdapi.sqc'      !Routines to Update Run Status
#include 'datetime.sqc'   !Routines for date and time formatting
#include 'number.sqc'     !Routines to format numbers
#include 'askaod.sqc'     !Ask As Of Date input
```

The Stdapi-Term procedure in Stdapi.sqc calls the Successful-Eoj procedure from EOJ.sqc which updates the run status to 'Successful'

The Stdapi-Init procedure in Stdapi.sqc call Define-Pracs-Vars and Get-Run-Control-Parms to initialize API variables, gets control parameters and updates the run status to 'Processing'

STDAPI.SQC
includes all
necessary
API code

At the program start, the Stdapi-Init procedure is invoked. This procedure is a part of the PeopleSoft-delivered SQC file STDAPI.sqc. Stdapi-Init invokes two more procedures in turn. The first one, Define-Pracs-Vars, is located in PRCSAPI.sqc. Its job is to initialize all API variables. The second procedure, Get-Run-Control-Parms, determines whether the program is called from the Process Scheduler and, if so, promotes the run status from Initiated to Processing.

Let's see how the Get-Run-Control-Parms procedure knows that the program is invoked from the Process Scheduler. Take a look at the procedure source code shown in the following example:

```
!The Get-Run-Control-Parms procedure
Begin-Procedure Get-Run-Control-Parms
    Input $prcs_process_instance
    'Please press ENTER (Do not input a value)'
    if not isnull($prcs_process_instance)
        let #prcs_process_instance = to_number($prcs_process_instance)
        input $prcs_oprid 'Please press ENTER (Do not input a value)'
        let $prcs_oprid = upper($prcs_oprid)
        input $prcs_run_cntl_id 'Please press ENTER (Do not input a value)'
    else
        let #prcs_process_instance = 0
    end-if
    if #prcs_process_instance > 0
        let #prcs_run_status = #prcs_run_status_processing
        do Update-Pracs-Run-Status
        let #prcs_run_status = #prcs_run_status_successful
    end-if
end-procedure
```

The first Input command is used to check where the program was called from

As you can see, the procedure code begins with the Input command. If the program is invoked from the regular SQR dialog window (which usually happens during the program's testing) or from the SQR command line, the operator receives the prompt 'Please press ENTER (Do not input a value)'. After the operator presses the ENTER key, the \$prcs_process_instance variable remains set to NULL, and the procedure logic can easily detect this.

If the program is invoked from the Process Scheduler, the \$prcs_process_instance variable receives its value from the parameter list passed from the Process Scheduler. The parameter list, besides the Process Instance value, also includes the operator ID, and the process run ID. Figure 28.2 shows the Process Request Detail panel for the Status Report. For this particular program run, the Process Instance is equal to 4, the Operator ID is PS, and the Process Run ID is MY_STATUS_01.

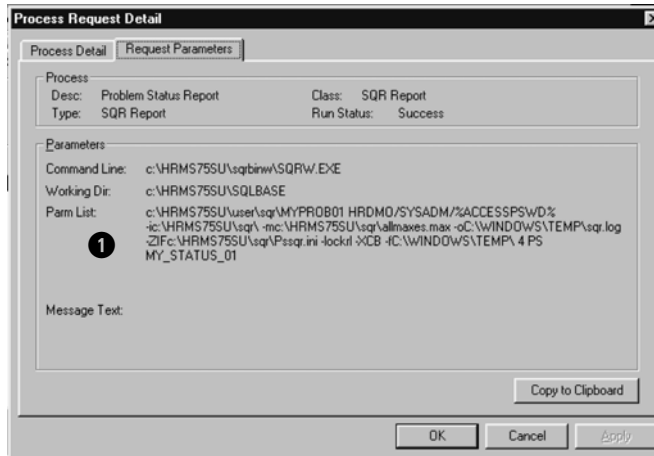


Figure 28.2 Run Control parameters passed to an SQR program

- ❶ The three Run Control parameters passed to an SQR program from the Process Scheduler.

Let's return to myprob02.sqr. At the end of the main section, the program calls the `Stdapi-Term` procedure, which is a part of `STDAPL.sqc`. The purpose of this procedure is to update the `PSPRSCRQST` table with process run status, the message parameters, and the return code. The chart in figure 28.3 will help you to figure out

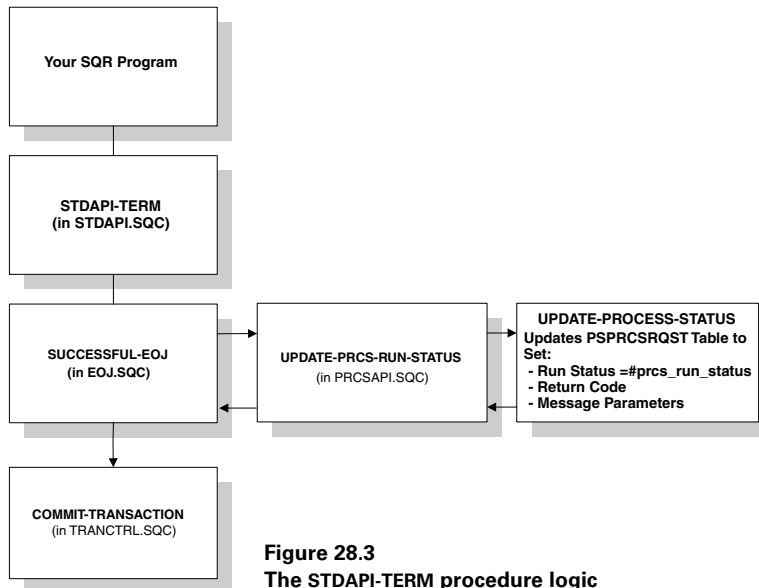


Figure 28.3
The `STDAPL-TERM` procedure logic

how the `Stdapi-Term` procedure communicates the program status to the Process Scheduler.

As you can see, the run status in the `PSPRCRQST` table is updated based on the `#prcs_run_status` variable value. This variable determines the run status, which you see on the Process Monitor panel.

28.2.2 Communicating errors back to the Process Scheduler

It is important to remember that, in case of an error, the value of the `#prcs_run_status` variable must be updated by the application program. In a normal run, PeopleSoft promotes the process run status in the following order: Queued, Initiated, Processing, Success.

Please note that `#prcs_run_status` is a numeric variable. It cannot be assigned the above text values directly. The `PRCSDEF.sqc` file includes a number of predefined numeric status variables that can be used to assign the right status value to the `#prcs_run_status` variable.

As soon as your program is scheduled to run, the Process Scheduler sets the run status on the Process Monitor to `Queued`. Next, if all parameters in the process definition are resolved and the system resources are available to run the process, the Process Scheduler changes the status to `Initiated`. If your program fails to get through the compilation stage, the status on the Process Monitor panel remains `Initiated` if your program runs on Client. If it runs on the Server, the status will be changed to `Error` by the SQR invocation script.

The `Stdapi-Init` procedure (which must be called in the beginning of every API Aware program) changes the status to `Processing` and updates the `PSPRCRQST` table. At this moment, you can see the status set to `Processing` on the Process Monitor panel. The `Stdapi-Init` procedure then sets the `#prcs_run_status` variable to `Success` (`#prcs_run_status_successful`) in the program memory only, but holds back from updating the `PSPRCRQST` table until either the `Stdapi-Term` or `SQL-Error` procedure is called. Therefore, you will still see the `Processing` status on the Process Monitor panel.

If your SQR program runs to the end, then calls the `Stdapi-Term` procedure as shown in figure 28.3, this procedure updates the process status to `Success`. In case of an error, it is your program's responsibility to call a PeopleSoft-delivered error-handling routine `SQL-Error` or code a similar logic in your program. Otherwise, the status on the Process Monitor either remains set to `Processing` if your program aborted during execution or, worse yet, is set to `Success` if the program ran to the end and called `Stdapi-Term` regardless of the error situation.

If your program uses a PeopleSoft-delivered error handling routine (part of which is shown in the following example), you do not have to worry about updating the API variables in an error situation. If, however, your program uses its own error-processing logic, the program must include a code to set all API variables to the proper values and

update the Process Request table PSPRCSRQST. Following is an example of the PeopleSoft-delivered SQL error-handling procedure:

<pre>!A part of the SQL Error !procedure in SQLEERR.SQC if #prcs_process_instance > 0 let #prcs_message_set_nbr = #prcs_msg_set_nbr let #prcs_message_nbr = #prcs_msg_nbr_sql_error let #prcs_run_status = #prcs_run_status_error let #prcs_rc = #sql-status let \$prcs_message_parm1 = \$sql-error let #prcs_continuejob = 0 do Rollback-Transaction if \$prcs_in_update_prcs_run_stat <> 'Y' do Update-Prcs-Run-Status do Commit-Transaction end-if end-if #endif VMS let #return-Status = 1 #endif stop</pre>	<div style="border-left: 1px solid black; padding-left: 10px;"> <p>This procedure is usually referenced in the On-Error parameter of the Begin-Sql</p> </div> <div style="border-left: 1px solid black; padding-left: 10px; margin-top: 20px;"> <p>Updating the API variables</p> </div> <div style="border-left: 1px solid black; padding-left: 10px; margin-top: 20px;"> <p>Updating the PSPRCSRQST table</p> </div>
---	---

As you can see, the error-processing logic in an API Aware program should include updating a set of API variables and calling the Update-Prcs-Run-Status procedure that updates the Process Request table for your program. After the Process Request table is updated, the Commit-Transaction function makes this table change permanent.

28.3 **CREATING A NEW PROCESS DEFINITION FOR AN API AWARE PROGRAM**

Since our program name has changed from myprob01.sqr to myprob02.sqr, a new process definition has to be created. Remember that a process definition must have exactly the same name as your SQR program. If we modify our SQR program without changing its name, we can just update the API Aware flag in the MYPROB01 process definition.

In our case, we create a new one. We repeat the same steps that we performed in chapter 27. Just remember that, this time our process must be marked as an API Aware process (figure 28.4).

After you save the panel in figure 28.4, the MYPROB02 process definition is created. It has the same characteristics as MYPROB01, except that the API Aware flag is now turned on. Also, we attached it to the same panel group, MY_PROB01. Will this present any problems? How will the Process Scheduler know which program to execute? Let's find out the answers to our questions by performing a simple test. Select Status Report from the Problem Tracking menu and press the Traffic Light tool bar button.

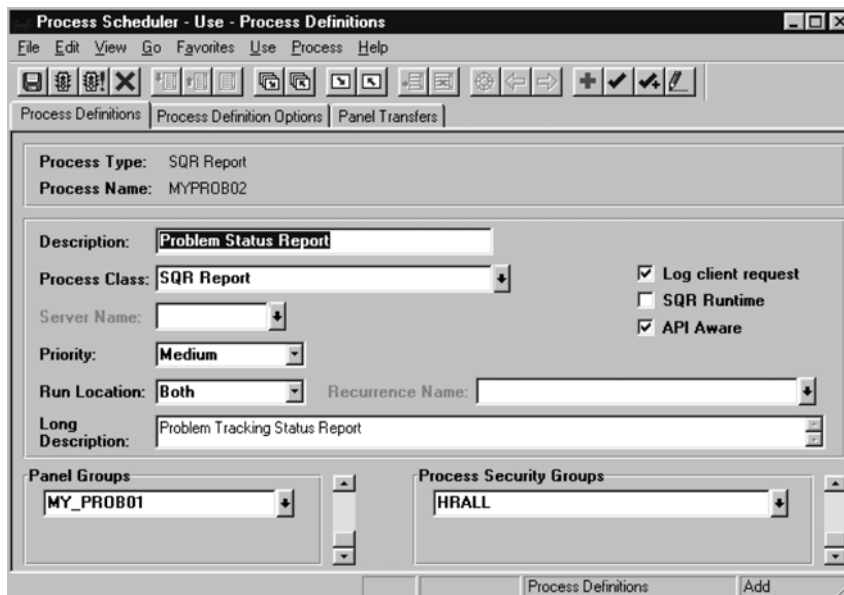


Figure 28.4 Creating a new process definition for an API Aware SQR program

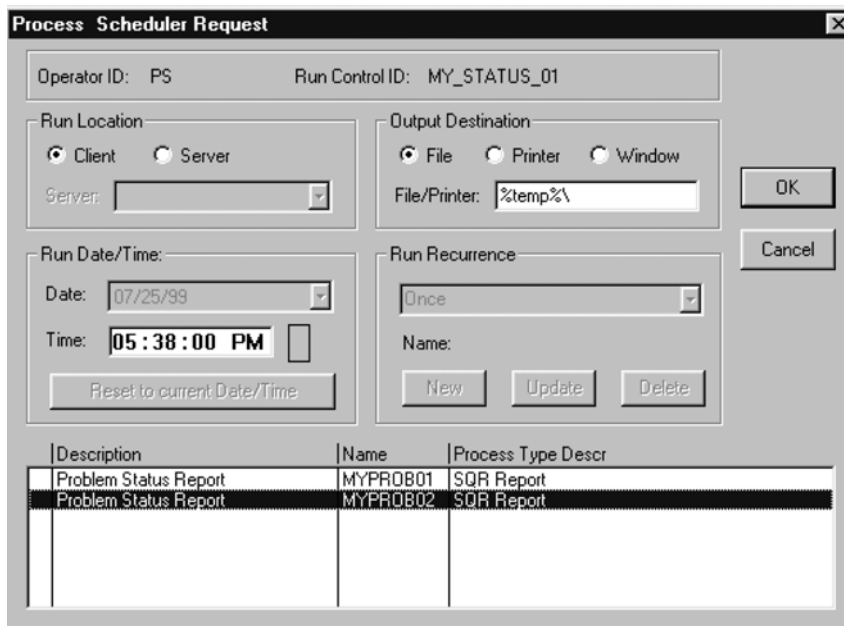


Figure 28.5 Two SQR programs are available to run from the Process Scheduler

As you can see from figure 28.5, two SQR programs are available now for execution from the Process Scheduler. Sometimes, it is a good and economical solution to have several programs under the same roof (attached to the same panel group). For example, if you have a detail report and a summary report, and you need to give your users a way to execute either one of the two, you can place them together. In this case, your users would have to highlight the process which they need to execute, and then press the OK button.

In our situation, however, there is no need to keep the first report. It was not designed to be executed from the Process Scheduler in the first place, and, therefore, the report did not have any API interface code to communicate with API functions. We can either disconnect this program from the panel group by deleting the panel group from its process definition, or, using our database-specific SQL tools, we can delete the obsolete process definition from the tools tables. Let's use the second option in order to keep our system clean.

28.3.1 Deleting the obsolete process definition

We have to use a trick here. Presently, PeopleSoft does not have any online tools available to delete obsolete process definitions from the database. Using our knowledge of the PeopleSoft system (tools) tables, and with the help of the native SQL, we can create a simple cleanup script:

```
delete from ps_prsdefn
where prcsname='MYPROB01' and prcstype = 'SQR Report';
delete from ps_prsdefngrp
where prcsname='MYPROB01' and prcstype = 'SQR Report';
delete from ps_prsdefnpl
where prcsname='MYPROB01' and prcstype = 'SQR Report';
delete from ps_prsdefnxfer
where prcsname='MYPROB01' and prcstype = 'SQR Report';
delete from psprcsrqst
where prcsname='MYPROB01' and prcstype = 'SQR Report';
delete from pspnlfield
where prcsname='MYPROB01' and prcstype = 'SQR Report';
```

After the Delete script is executed, log out from your PeopleSoft system, delete your cache files, and log back on to PeopleSoft.

To verify that the script we executed actually gave us the result we expected, we repeat the steps again to bring up the process request with Problem Status report. We can see that this time only one report is available for execution (figure 28.6)

Description	Name	Process Type Descr
Problem Status Report	MYPROB02	SQR Report

Figure 28.6 Only one report is available now for execution

We can click on the OK button and execute our report now, but, if you remember, we have one more important problem to address. Our program does not currently accept any parameters from the online panel that we created. Let's first discuss what tools PeopleSoft offers to simplify the task of obtaining the input parameters, then implement this in our program.

28.4 **EXERCISE 3: ACCEPT THE AS OF DATE AND PROBLEM STATUS PARAMETERS FROM AN ON-LINE PANEL**

Our task now is to modify MYPROB02.sqr so that it knows when it is being executed from the PeopleSoft online panel and accepts the parameters without further prompting. The program should also retain the functionality of the Input prompt when it is executed outside the Process Scheduler.

We have already done most of the work: we've developed the Run Control record and panel and made sure that the online part of this project is working properly. We were able to enter our parameters and to save them in the record. Let's do the rest now.

28.4.1 **Using application-specific SQC files to obtain input parameters**

PeopleSoft delivers a number of application-specific SQC files that are used to read input parameters from Application Run Control records. Usually, two SQC files are involved in reading the parameters: one file selects the input parameters, while

another one formats the selected parameters and moves them to the designated SQR program variables. You can either use the PeopleSoft-delivered SQC files or develop your own, depending on the parameters your SQR program needs to accept.

Let's first learn how PeopleSoft-delivered SQR programs work with input parameters. Later, you will learn how to use a similar approach in your program. Since you have already become familiar with the PeopleSoft-delivered Years of Service report, let's examine how this program works. We know that this program accepts two input parameters: As Of Date and Years of Service.

28.4.2 How the Years of Service program accepts its input parameters

The name of the program that generates the Years of Service report is PER003.sqr. If you open the PER003.sqr, and scroll down to the end of the program code, you can see that it uses the following SQC files:

```
!SQC files that are used to obtain input parameters in the
!Years of Service program
#include 'hrrnctl1.sqc' !Get Run Control parameter values
#include 'hrgetval.sqc' !Get values mask routines
#include 'askaod.sqc' !Ask As Of Date input
#include 'asksrvyr.sqc' !Years Of Service input
```

This is how the input parameter read section of PER003.sqr appears:

```
!Procedures used in reading input parameters in PER003.sqr
begin-procedure Init-Report

    move 'PER003' to $ReportID
    do Delete-Worktable
    do Stdapi-Init
    if $prcs_oprid=''
        display ''
        display 'REPORT CAN NOT BE EXECUTED OUTSIDE OF PEOPLESOFT,PLEASE USE
PROCESS SCHEDULER.'
        display ''
        goto last1
    end-if

    do Sqr-Param

    if $prcs_process_instance = ''
        do Ask-As-Of-Date
        do Ask-Years-Of-Service
    else
        do Select-Parameters
    end-if
    do Init_Printer
    do Init_Report_Translation ($ReportID, $language_cd)
    do Append_Report_Translation ('HR')
```

← **Check to see if run from the Process Scheduler. If yes, call Select-Parameters.**

```

last1:

end-procedure

begin-procedure Get-Values
    let $language_cd = $PRCS_LANGUAGE_CD
    do Get-As-Of-Date
    do Get-Years-Of-Service

end-procedure

```

This procedure is called from the Select-Parameters procedure.

In the previous subchapter, we found that the `Stdapi-Init` procedure initializes API variables and obtains the Process Scheduler command-line parameters (if any). Next, the program determines the method of its invocation. The program may be initiated by the Process Scheduler or invoked some other way (submitted via the SQR dialog box, executed from the command line, or called by another application). Based on this check result, the program calls the proper subroutine to obtain the application-specific input parameters.

If the program is not run under the Process Scheduler, the `$prcs-process-instance` variable remains empty and the regular SQR Input command is used in the `Ask-As-Of-date` and `Ask-Years-Of-Service` subroutines to read the input parameters from user input. The `Ask-As-Of-date` code is located in the `ASKAOD.sqc` file, and the `Ask-Years-Of-Service` is located in the `ASKSRVYR.sqc` file.

If the program is invoked by the Process Scheduler, the `$prcs-process-instance` variable is assigned the process instance number value, and the `Select-Parameters` subroutine is called to retrieve the input parameters from a specific application Run Control table. In terms of the Years of Service report, the program is designed to work with the Run Control table named `PS_RUN_CNTL_HR`, but the procedure logic is a typical example of the communication between an SQR program and a PeopleSoft online panel.

Let's examine the `Select-Parameters` procedure. The procedure code is located in the `HRRNCTL1.sqc` file:

```

!A typical input parameters read procedure in HRRNCTL1.SQC
begin-procedure select-parameters
BEGIN-SELECT
RUN_CNTL_HR.OPRID
RUN_CNTL_HR.RUN_CNTL_ID
RUN_CNTL_HR.ASOFDATE
RUN_CNTL_HR.FROMDATE
RUN_CNTL_HR.THRUDATE
RUN_CNTL_HR.CALENDAR_YEAR
RUN_CNTL_HR.SERVICE_YEARS
RUN_CNTL_HR.AD_STEP
RUN_CNTL_HR.AD_STEP_ENTRY_DT
RUN_CNTL_HR.AD_COMPRATE

```

```

RUN_CNTL_HR.AD_HOURLYRT
RUN_CNTL_HR.AD_MONTHLYRT
RUN_CNTL_HR.AD_ANNUALRT
!...
!...
RUN_CNTL_HR.AD_CHANGEAMT
RUN_CNTL_HR.AD_CHANGEPCT
RUN_CNTL_HR.EEO_REPORT_TYPE
  do Get-Values
from PS_RUN_CNTL_HR RUN_CNTL_HR
where RUN_CNTL_HR.OPRID = $prcs_oprid
  and RUN_CNTL_HR.RUN_CNTL_ID = $prcs_run_cntl_id
end-select
end-procedure

```

If your program includes the HRRNCTL1.sqc file, a procedure named Get-Values should be coded withing your program.

In the previous procedure developed by PeopleSoft, the application-specific input parameters are selected from the PS_RUN_CNTL_HR table for a given combination of operator ID (\$prcs_oprid) and Run Control ID (\$prcs_run_cntl_id). As you learned in the previous chapter, these two variables come from the Process Scheduler parameter list. An important and not-to-be-missed part of the Select-Parameters procedure is a call to the Get-Values procedure. This procedure moves and edits the selected input parameter values to the designated variables in an SQR program. If your program uses the HRRNCTL1.sqc file, the name of the input parameter edit subroutine must be Get-Values. If you code the input parameter retrieval logic yourself, the name of this subroutine (if any) can be different.

In the Years of Service report, a subroutine named Get-Values is a part of the Per003.sqr code. You can see this subroutine in our previous example explaining procedures used to read input parameters. Because the Per003.sqr program accepts the Language Code and two application-specific parameters, As Of Date and Years of Service, the Get-Values subroutine in this case is simple. It moves the Language Code value to its designated program variable and then calls the Get-As-Of-Date and Get-Years-Of-Service procedures to format these two variables and to move them to their respective designated variables:

```

begin-procedure Get-Values
  let $language_cd = $PRCS_LANGUAGE_CD
  do Get-As-Of-Date
  do Get-Years-Of-Service
end-procedure

```

28.4.3 Accepting input parameters in your SQR program

Now that you have learned how a PeopleSoft-delivered program retrieves its application-specific input parameters, let's apply this knowledge to the applicable Problem Status report.

Our program has two input parameters: As Of Date and Problem Status. We want to create our own include (SQC) files, which will help us in selecting and reformatting input parameters for our program.

28.4.4 Creating your own SQC files

Please note that you do not have to place all input parameter retrieval and reformatting logic into SQC files. This is just a convenient and modular way to read the input parameters. It also gives you an advantage when you want to re-use this code for other programs. Another way of working with your input parameters is to place this logic directly in the application program.

In order to create new application-specific SQC files, we will be using our preferred technique of cloning the existing SQC files. You already know that you need to have one SQC file to select the input parameter values from the appropriate Run Control record and another one to format the selected values and move them to designated variables in your program. Bearing in mind that your application program should retain an ability to be executed from either the SQR dialog box or the command line, you must also provide the code to prompt the user for input parameters.

28.4.5 Creating an SQC file to select parameters from the Run Control record

To create a new input parameter retrieval SQC file, we'll clone the existing HRRNCTL1.sqc file. Let's bring this file in, save it as MYRUNCTL.sqc and change it to make it work with the MY_RUN_CNTL Run Control record:

```
!The modified Select-Parameters procedure in MYRUNCTL.sqc
Begin-Procedure Select-Parameters
Begin-Select
OPRID
RUN_CNTL_ID
ASOFDATE
MY_PROBLEM_STATUS
    Do Get-Values
From    PS_MY_RUN_CNTL
Where OPRID = $prcs_oprid
And     RUN_CNTL_ID = $prcs_run_cntl_id
End-Select
End-Procedure
```

28.4.6 Creating an SQC file to format selected input parameters

We use the existing HRGETVAL.sqc file as a basis when creating the new input parameter formatting file MYGETVAL.sqc. All you need to do is delete the commands that format unused input parameters, and add logic to format your parameters. The changed program, MYGETVAL.sqc, is listed as follows:

```

! *****
! MYGETVAL.SQC:
! *****
Begin-Procedure Get-As-Of-Date
! *****
    Let $AsOfDate = RTRIM(&Asofdate, ' ')
    If $AsOfDate = ''
        Move $AsOfToday to $AsOfDate
    End-if
End-Procedure
! *****
Begin-Procedure Get-Problem-Status
! *****
Let $Problem_Status = RTRIM(&MY_PROBLEM_STATUS, ' ')
End-Procedure

```

As you can see, the modified program includes the Get-As-Of-Date procedure for the As Of Date column variable. In addition, a new Get-Problem-Status procedure is added to get the additional parameter, MY_PROBLEM_STATUS.

After your SQC file is created, it should be saved in the directory specified in the Configuration Manager panel under the Process Scheduler tab, in the SQR Flags parameter for Client program execution.

28.4.7 Integrating the SQC files with your program

Let's make a few modifications to the Problem Status report to make it work with the newly created Run Control record and to include the new SQC files MYRUNCTL.sqc and MYGETVAL.sqc. This time we list the entire program (listing 28.1):

Listing 28.1

```

!MYPROB02.SQR
!Problem Status Report

#include 'setenv.sqc'

#define problem_status_len 10
#define project_descr_len 30
#define date_len 10
#define priority_len 8
#define user_name_len 20
#define responsible_name 20
#define col_sep 2

! *****
Begin-Setup
! *****
Load-Lookup Name=Projects
    Rows = 500
    Table = PS_MY_PROJECT_TBL

```

```

        Key    = MY_PROJECT_ID
        Return_Value=Descr

Load-Lookup Name=Users
    Rows    = 1000
    Table   = PS_MY_USER_TBL
    Key     = MY_USER_ID
    Return_Value=Name

End-Setup

!*****
Begin-Heading 7
!*****
print 'Problem Status Report' (1,1) Center

page-number                (0,100)  'Page No.  '
print 'Run Date '          (+1,100)
print $ReportDate          ( )
print 'Run Time '          (+1,100)
print $ReportTime          ( )

Print 'Problem Status: '   ( ,1)
Print $Stat                ( )

print '='                  (+1,      1,      125) fill
print 'Project Description ' (+1,      1, {project_descr_len} )
print 'Incident '          ( ,+{col_sep}, {date_len} )
print 'Priority '          ( ,+{col_sep}, {priority_len} )
print 'User Name '         ( ,+{col_sep}, {user_name_len} )
print 'Responsible '       ( ,+{col_sep}, {responsible_name} )
print 'Close '             ( ,+{col_sep}, {date_len} )

print ' '                  (+1,      1, {project_descr_len} )
print ' Date '             ( ,+{col_sep}, {date_len} )
print ' '                  ( ,+{col_sep}, {priority_len} )
print ' '                  ( ,+{col_sep}, {user_name_len} )
print 'To Resolve '        ( ,+{col_sep}, {responsible_name} )
print 'Date '              ( ,+{col_sep}, {date_len} )
print '='                  (+1,      1,      125) fill

End-Heading

!*****
Begin-Program
!*****
do Init-DateTime
do Init-Number
Do Init-Report
Do Main
Do Stdapi-Term

```

End-Program

!*****

Begin-Procedure Init-Report

!*****

Do Stdapi-Init

If \$prcs_process_instance = ''

Do Ask-Input-Parameters

Else

Do Select-Parameters

End-if

Do Build-Where

Do Load-Xlats

End-Procedure

Check to see if run from the Process Scheduler; If yes, call Select- Parameters; otherwise, call Ask-Input-Parameters

Located in MYRUNCTL.sqc

!*****

Begin-Procedure Get-Values

!*****

Do Get-As-Of-Date

Do Get-Problem-Status

Call the Get-As-Of-Date and Get-Problem-Status procedures from MYGETVAL.sqc

End-Procedure

!*****

Begin-Procedure Load-Xlats

!*****

Let \$Where_Xlat1 = 'FIELDNAME='MY_PRIORITY''

|| ' and X.EFFDT = (Select max(Effdt) from XLATTABLE '

|| 'Where Fieldname=X.Fieldname And FieldValue=X.FieldValue'

|| ' And Effdt <= Sysdate and Language_Cd = 'ENG') '

Load-Lookup Name=Priority

Rows = 10

Table = 'XLATTABLE X'

Key = FIELDVALUE

Return_Value=XLATSHORTNAME

Where=\$Where_Xlat1

Let \$Where_Xlat2 = 'FIELDNAME='MY_PROBLEM_STATUS''

|| ' and S.EFFDT = (Select max(Effdt) from XLATTABLE '

|| 'Where Fieldname=S.Fieldname And FieldValue=S.FieldValue'

|| ' And Effdt <= Sysdate) '

Load-Lookup Name=Status

Rows = 20

Table = 'XLATTABLE S'

Key = FIELDVALUE

Return_Value=XLATSHORTNAME

Where=\$Where_Xlat2

End-Procedure


```

!*****
Begin-Procedure Ask-Input-Parameters
!*****
!Get User's Input

Do Ask-As-Of-Date                               !in askaod.sqc

Let #Input=1
While #Input = 1
    Input $Problem_Status Type=Char 'Please Enter Problem Status(1=Initi-
ated, 2=Assigned, 3=Progress, 4=Testing, 5=Resolved,6=Void) or press Enter
for All' Status=#Input_Status
    If $Problem_Status = ''
        Let #Input = 0
    Else
        If $Problem_Status > '0' and $Problem_Status < '7'
            show 'Problem Status Entered = ' $Problem_Status
            Let #Input = 0
        Else
            Show 'Invalid Input, Re-Entry Required'
        End-If
    End-If
End-While

End-Procedure

!*****
Begin-Procedure Build-Where
!*****
!Build Where Clause based on user's Input
If $Problem_Status = ''
    Let $Where_status = ''
Else
    Let $Status=Rtrim($Problem_Status,' ')
    Let $Where_status = 'And A.My_Problem_Status = ' || ' ' || $Status || ' '
    Show $Where_status
End-If

End-Procedure

!*****
Begin-Procedure Main
!*****
Begin-Select
A.My_Problem_Status      ()      on-break Print=Never After=Page-Break
Save=$Status_Cur
A.My_Project_ID
A.Incident_DT
A.My_Priority
A.My_User_ID
A.My_Problem_Tracker
A.Close_Dt

```

**Call PeopleSoft-delivered function
to get As-Of-Date prompt.**



```

        Do Print-Line
From PS_MY_PROBLEM_TRKG A
Where A.Incident_Dt <= $AsOfDate
[$Where_status]
order by A.My_Problem_Status
End-Select

End-Procedure

!*****
Begin-Procedure Print-Line
!*****
Lookup Projects &A.My_Project_ID $Descr
Print $Descr          (+1,          1, {project_descr_len} )
Print &A.Incident_DT   ( ,+{col_sep}, {date_len}           )
Lookup Priority &A.My_Priority $Priority_Descr
Print $Priority_Descr  ( ,+{col_sep}, {priority_len}        )
Lookup Users &A.My_User_ID $User_Name
Print $User_Name      ( ,+{col_sep}, {user_name_len}       )
Lookup Users &A.My_Problem_Tracker $Problem_Tracker_Name
Print $Problem_Tracker_Name ( ,+{col_sep}, {responsible_name} )
Print &A.Close_Dt      ( ,+{col_sep}, {date_len}           )

End-Procedure

!*****
Begin-Procedure Page-Break
!*****
Lookup Status $Status_Cur $Stat
new-page
End-Procedure

!*****

#include 'stdapi.sqc'      !Routines to Update Run Status
#include 'datetime.sqc'    !Routines for date and time formatting
#include 'number.sqc'      !Routines to format numbers
#include 'askaod.sqc'      !Ask As Of Date input
#include 'myruncntl.sqc'    !Get Run Control parameters
#include 'mygetval.sqc'     !Format Run Control parameters

```

The custom SQC files are added to the program.

In listing 28.1, the code of the Status Report was changed to include the MYRUNCNTL.sqc and MYGETVAL.sqc files. Also, we added a code to the Init-Report and Get-Values procedures to reflect the new SQC functionality.

28.5 TESTING YOUR CHANGES

Before we start testing, let's list all the modifications that were made to make the Problem Status report API Aware and to enable it to accept input parameters from the Process Scheduler:

- The Stdapi-Init and Stdapi-Term procedure calls were added to the MYPROB02.sqr program. The STDAPL.sqc include file was incorporated into our program.
- The MYRUNCTL.sqc file was created to select input parameters from the MY_RUN_CNTL record.
- The MYGETVAL.sqc file was created to format the selected parameters and to move them to the designated program variables.
- The MYRUNCNTL.sqc and MYGETVAL.sqc files were included in the program.
- The Init-Report and Get-Values procedures in MYPROB02.sqr were modified to call the new Select-Parameters, Get-As-Of-Date, and Get-Process-Status procedures.

Let's execute our program, then verify the program output report.

Since we already created a Run Control ID, MY_STATUS_01, we can re-use it. Once displayed, we can change its input parameters. Let's enter 08/03/1999 in the As Of Date field, and select Assigned as our problem status (figure 28.7).

Navigation: Go → Problem Tracking → Report → Status Report → Update/Display

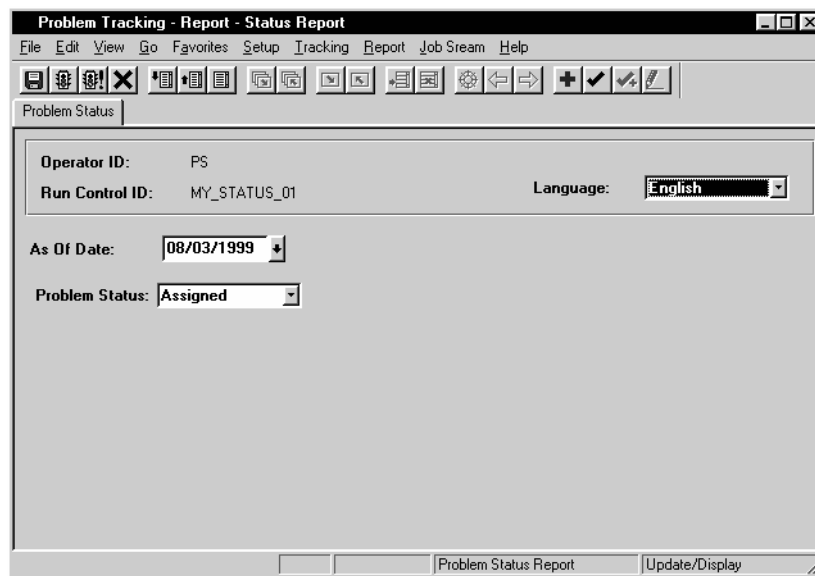
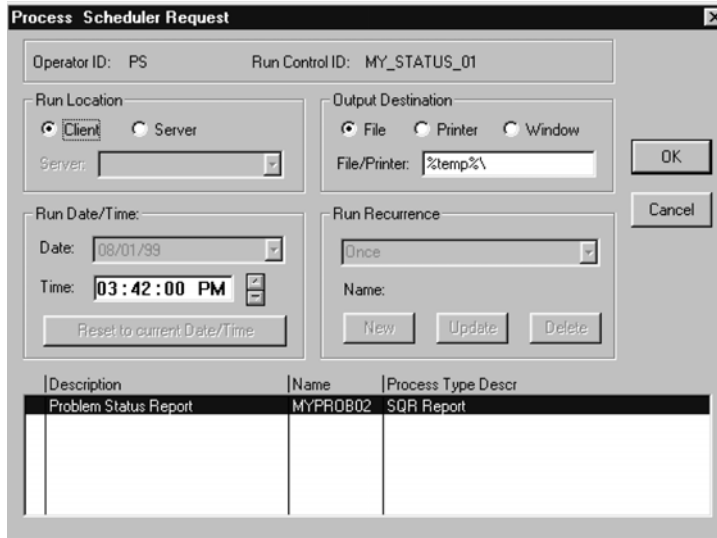


Figure 28.7 Executing the Problem Status report

When all parameters are entered, click on the traffic light to bring in the Process Request panel.



Operator ID: PS Run Control ID: MY_STATUS_01

Run Location: ☒ Client ☐ Server
 Server:

Output Destination: ☒ File ☐ Printer ☐ Window
 File/Printer: %temp%\

Run Date/Time: Date: 08/01/99 Time: 03:42:00 PM
 Reset to current Date/Time

Run Recurrence: Once
 Name: New Update Delete

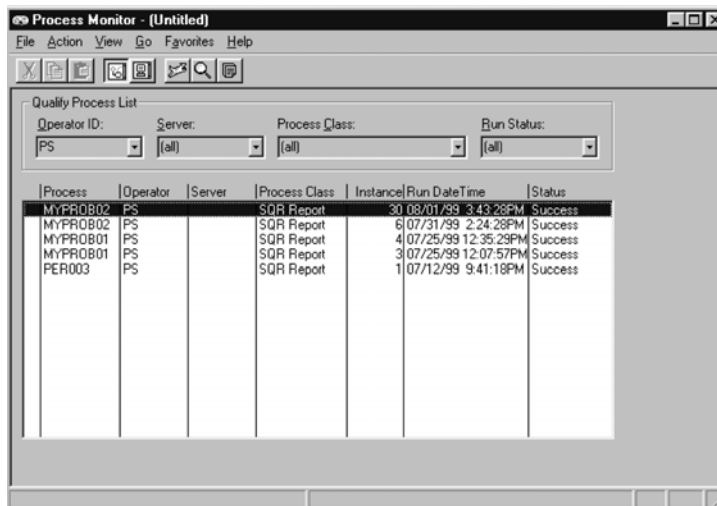
Description	Name	Process Type Descr
Problem Status Report	MYPROB02	SQR Report

OK Cancel

Figure 28.8 Process Request for MYPROB02.sqr

Let's click on the OK button and run the program. Since we made our program API Aware, it should send its process status to the Process Scheduler. The Process Monitor screen helps us to see the status of our program.

Navigation: Go →PeopleTools →Process Monitor



Process Monitor - (Untitled)

File Action View Go Favorites Help

Quality Process List

Operator ID: PS Server: (all) Process Class: (all) Run Status: (all)

Process	Operator	Server	Process Class	Instance	Run Date/Time	Status
MYPROB02	PS		SQR Report	30	08/01/99 3:43:28PM	Success
MYPROB02	PS		SQR Report	6	07/31/99 2:24:28PM	Success
MYPROB01	PS		SQR Report	4	07/25/99 12:35:29PM	Success
MYPROB01	PS		SQR Report	3	07/25/99 12:07:57PM	Success
PER003	PS		SQR Report	1	07/12/99 9:41:18PM	Success

Figure 28.9 Examining the status of our program execution

Our program executed successfully. Let's verify the output report. First, double-check the destination of our output report. In order to find this, double-click on the process name of our process (MYPROB02) in the Process Monitor screen, then go to the second tab, Request Parameters.

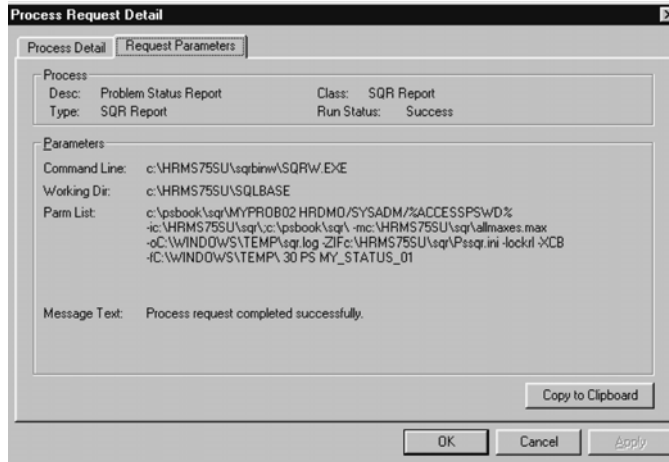


Figure 28.10
Verifying the output file destination

Take a look at the Parm list in figure 28.10. The report output file is specified with the `-f` flag. Therefore, we look for our .lis file in the `c:\windows\temp` directory.

-
- TIP** An SQR program produces its output file only if there were at least one detail output record. If your program ran to `success`, but you cannot find your output report, check the following:
- Are you looking at the right directory?
 - Are your input parameters correctly specified?
 - Do you have data in the database?
 - Is your selection criteria correct?
-

In our case, we received the report, shown in figure 28.11.

As you can see, our program ran successfully and produced the report which shows only the assigned incidents. Let's do one more test and see how our program handles the situation when we do not enter a specific problem status. We want to print all incidents entered into the system as of 08/03/99 in our report.

myprob02.lis - Notepad

Problem Status Report						Page No. 1
Problem Status: Assigned						Run Date 08/01/1999
						Run Time 15:43:31
Project Description	Incident Date	Priority	User Name	Responsible To Resolve	Close Date	
General Ledger Customizations	1998-06-01	Medium	Bitnap,Hector	Tahari,Peter		
Union Mass Change	1999-07-14	High	Tahari,Peter	Barnie,John		
Department Security	1999-07-10	High	Mentor,Bill	Kaplan,Joseph		

Figure 28.11 The output of MYPROB02.sqr report

Let's enter our parameters (figure 28.12).

Problem Tracking - Report - Status Report

File Edit View Go Favorites Setup Tracking Report Job Stream Help

Problem Status

Operator ID: PS

Run Control ID: MY_STATUS_01

Language: English

As Of Date: 08/03/1999

Problem Status:

Problem Status Report Update/Display

Figure 28.12 Entering an empty Problem Status to select all incidents as of 08/03/1999 date

After our program executes, let's again examine the output report: MYPROB02.lis.

Problem Status Report					Page No. 1
Problem Status: Initiated					Run Date 08/01/1999
					Run Time 17:50:34
Project Description	Incident Date	Priority	User Name	Responsible To Resolve	Close Date
HR Customizations	1999-07-01	Low	Barnie, John	Mentor, Bill	

Problem Status Report					Page No. 2
Problem Status: Assigned					Run Date 08/01/1999
					Run Time 17:50:34
Project Description	Incident Date	Priority	User Name	Responsible To Resolve	Close Date
General Ledger Customizations	1998-06-01	Medium	Bitmap, Hector	Tahari, Peter	
Union Mass Change	1999-07-14	High	Tahari, Peter	Barnie, John	
Department Security	1999-07-10	High	Mentor, Bill	Kaplan, Joseph	

Figure 28.13 Problem Status report output that includes all problems grouped by Problem Status

As you can see from figure 28.13, the program produced several pages of the report. Therefore, our break logic as well as our parameter selection logic works. To confirm the test results, it is a good habit to use your native SQL tool and select records from the database to verify your report output. We can also use the online panels of our Problem Tracking application, and compare the report results. In addition, the report has to be tested on the Server to make certain that it runs correctly on both platforms. You should never assume that if your report works correctly on one platform, it runs without problems on another.

KEY POINTS

- 1** An API Aware process is a process that updates the Process Request table (PSPRCRQST) with the process run status (*Error*, *Success*, and so on), completion code, message set, and message number.
- 2** To accept input parameters from PeopleSoft online panels, you can either use the existing PeopleSoft-delivered SQC files or develop your own, depending on the parameters your SQR program needs to accept.
- 3** Your program should support both types of input parameter retrieval logic: retrieving the parameters from the Process Scheduler, and accepting them from the SQR Dialog Box or the command line.
- 4** Usually, there are two SQC files involved in accepting program input parameters from a PeopleSoft online panel. One file should contain a procedure to select all required fields from the proper Run Control record. Another one should include procedures to edit the selected fields and place them into designated SQR variables.
- 5** Your SQR program must be changed to include the proper SQC files and a code to call the input parameter retrieval procedures.
- 6** A Run Control panel which contains all the input parameters should be developed, or an existing panel should be used or customized.
- 7** The changed SQR program must be thoroughly tested on both Client and Server to make sure that the input parameters are passed and accepted correctly.



CHAPTER 29

Implementing security in SQR

- | | |
|---|---|
| 29.1 Overview of the PeopleSoft security layers 659 | 29.4 Incorporating Row-Level security in SQR 668 |
| 29.2 Row-level security in PeopleSoft online applications 660 | 29.5 Using Run Control records for SQR security 672 |
| 29.3 Preventing an SQR program from executing outside the Process Scheduler 666 | |

29.1 OVERVIEW OF THE PEOPLESOFT SECURITY LAYERS

Most of PeopleSoft-delivered applications work with important and sensitive information. Therefore, implementing and maintaining data security is usually a high priority task.

Before we start a discussion of different methods of implementing security in your SQR programs, let's review the online security functionality provided by PeopleSoft. Please be aware that, while comprehensive online Security Administration is not in the scope of this book (refer to PeopleSoft technical documentation for details), we will show you in great detail how to implement security in your SQR programs.

As we already discussed in the previous parts of this book, PeopleSoft provides you with layers of security to help protect your data from unauthorized access.

When accessing a PeopleSoft application in a networked environment, you have to pass through network security, database security, and PeopleSoft online security.

Network security typically includes the following components:

- an assigned ID and password for user verification
- an authorized sign-on time
- file access rights

In order to execute an SQR program, for example, users need to have appropriate access to the directory where the SQR executable resides and to the delivered or custom SQR programs. They also need to have access to the produced reports and input/output files (if any).

Database security is comprised of RDBMS (Relational Database Management System) security and PeopleSoft online security, which work together. The RDBMS security typically controls the database logon, database tables access and manipulations, and system administration activities.

PeopleSoft Online security includes *Operator Security* and *Object Security*. PeopleSoft provides you with utilities to maintain these two types of security. (Please refer to chapter 3.)

Row-Level security is used to control the user's access to specific rows of data from the database tables. PeopleSoft delivers applications with row-level security. PeopleSoft uses security search view records to provide online row-level security.

The *Field-Level security* can be implemented by using PeopleCode. For example, if you allow your operator to see a certain panel but would like to hide some fields in this panel, you can add logic to check the Operator ID, and either hide or show sensitive fields.

PeopleSoft also provides you with powerful tools to manage and enhance security based on your specific needs and applications. Using PeopleTools, you can design your own *Row-Level* and *Field-Level* security.

In the following subchapters, we are going to show you how to implement Row-Level security in batch SQR programs. Let's see first how Row-Level security works in PeopleSoft online programs. We will use similar approaches in batch SQR programs.

29.2 ROW-LEVEL SECURITY IN PEOPLESOFT ONLINE APPLICATIONS

PeopleSoft delivers a special way of controlling online access to your specific data rows by the means of security search records. These records are, in fact, regular SQL views designed with security in mind. You can either design your own security search records or use the PeopleSoft-delivered ones. After a view is created to be used as a

security search record, PeopleTools lets you attach the view to the corresponding PeopleSoft table and panel group. PeopleSoft delivers different security search mechanisms based on the application. For example, the built-in Department security is delivered with PeopleSoft HRMS package, while PeopleSoft Financial applications secure financial transactions by business units and ledgers. We will be using the HRMS application to review the online security features delivered by PeopleSoft.

If you look at any of your application core record's properties, you can see how PeopleSoft attaches these views to the record definition. Let's display, for example, the property window of the JOB record.

After the record definition is displayed, press ALT/ENTER to display the record's properties, and switch to the Use tab (figure 29.1).

Navigation: GO →Application Designer →Open →Record →JOB

The screenshot shows the 'Record Properties' dialog box with the 'Use' tab selected. The 'Set Control Field' dropdown is empty. Under 'Record Relationships', 'Parent Record' is set to 'EMPLOYMENT', 'Related Language Record' is empty, and 'Query Security Record' is set to 'EMPLMT_SRCH_GBL'. Under 'Record Audit', 'Record Name' is empty, and the 'Audit Options' section has four unchecked checkboxes: 'Add', 'Change', 'Selective', and 'Delete'. 'OK' and 'Cancel' buttons are at the bottom.

Figure 29.1
EMPLMT_SRCH_GBL is a query security record for the JOB record

As you can see from figure 29.1, the search view EMPLMT_SRCH_GBL is specified as a query security record for the JOB record. This lets the PeopleSoft system know what security record should be used to restrict the user's query access to the Job table.

Different tables may have the same or different query security records, depending on the table structure and the key fields it contains. For example, the PERSONAL_DATA table has the PERS_SRCH_QRY as its query security record, and the EMPLOYMENT table has the same search record as JOB. If you look at these view definitions, you can see that they are designed to restrict operator access to employee rows based on the department security that was set up for an operator.

29.2.1 Row-Level security in the PeopleSoft Query tool

In chapter 24, we discussed the nature of the EMPLMT_SRCH_US security record. If you open the EMPLMT_SRCH_GBL record in the Application Designer, you can see that this record is similar to the US record. In fact, the security mechanism is absolutely the same. Since we already learned how the record is built (and even took a brave attempt to modify it), we will show here how it's designed to work with the online QUERY tool.

Let's go to the Query tool and select the JOB record.

Navigation: Go → PeopleTools → Query

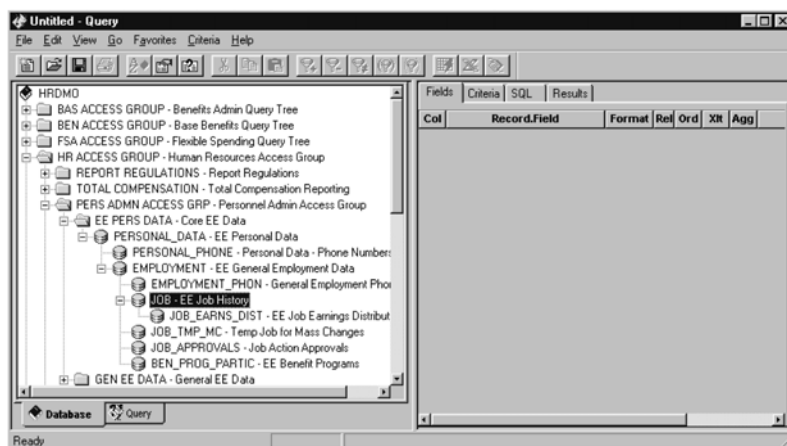


Figure 29.2 Selecting data from the JOB table via the Query tool

Double-click on the JOB record and select a few fields from this record. Take a look at the SQL statement by clicking on SQL tab on the right side of the Query panel.

What we see in figure 29.3, is an SQL statement that PeopleSoft generated on our behalf, based on the requirements it received from us. It contains the selection of the PS_JOB table columns that we specified. In addition, it joined the PS_JOB table with the PS_EMPLMT_SRCH_GBL table.

Now we can see how the security view limits the PS_JOB table row selection based on the previously defined security level. The join returns only rows defined in the security view PS_EMPLMT_SRCH_GBL. PeopleSoft joins the PS_JOB table with the Query Security record that we specified in the JOB's record definition.

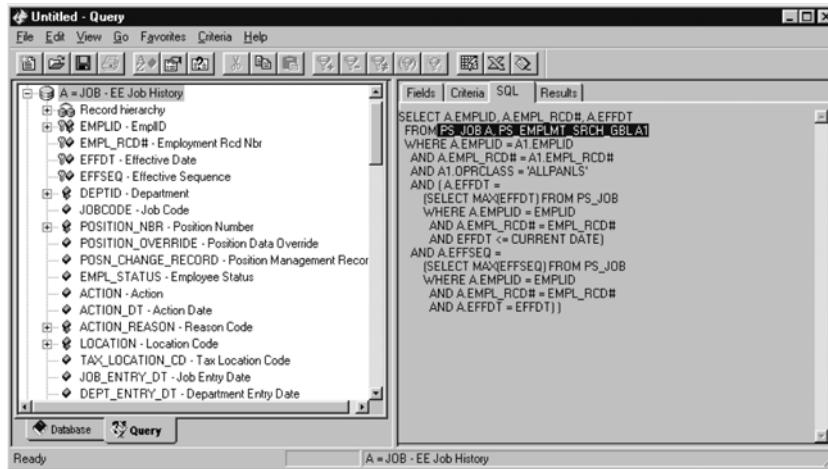


Figure 29.3 PeopleSoft generated SQL statement

What if we have to select data from two or more tables? Let's take a look at the SQL statement that PeopleSoft generates when we ask it to join our JOB table with the PERSONAL_DATA table (figure 29.4).

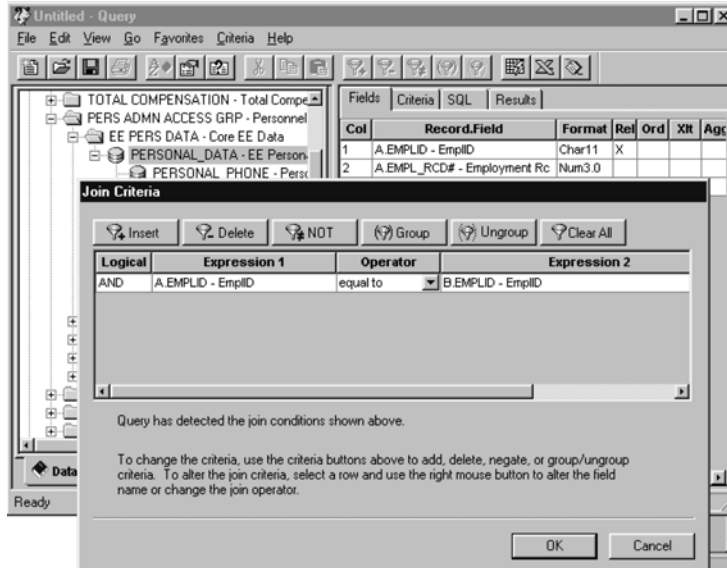


Figure 29.4
Adding the
PERSONAL_DATA
table to the Query

We examine the SQL statement now by switching to the SQL tab.

As you can see from the SQL statement in figure 29.5, in addition to joining PS_JOB with the PS_PERSONAL_DATA table, PeopleSoft also joins these two tables with the PS_EMPLMT_SRCH_GBL and PS_PERS_SRCH_QRY security search views. Therefore, when two records are joined with different security search records, PeopleSoft automatically joins the corresponding security views.

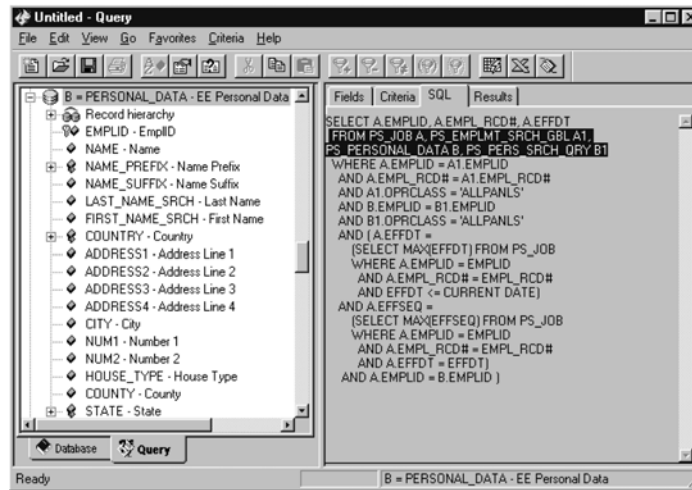


Figure 29.5 The PERSONAL_DATA table is joined with the Job table

Let's see one more example. Suppose we want to add the EMPLOYMENT table to our query. This table (and you can easily verify it) has the same security search view as the JOB table. Will PeopleSoft add the EMPLOYMENT table along with its security view or will it take the table alone? Let's check this out.

If you look again at the SQL statement that PeopleSoft generated, you see that the EMPLOYMENT table is joined without an additional security view. PeopleSoft is smart enough to recognize that the EMPLMT_SRCH_GBL is already present in the query.

Now that we learned that PeopleSoft automatically joins its tables with their respective security views, it's clear that the user's selection is controlled based on the view definition and the security setup. As we mentioned in the beginning of this part, the goal of our Security overview is to show the PeopleSoft security functionality that is relevant to our task of implementing security in SQR. The Security administration is a complex topic. In our examples, we only showed you how PeopleSoft joins its tables with their security views. The views themselves would never work without the proper Department security setup and administration. Each operator's security must be set up in conjunction with their business needs in order to gain appropriate access.

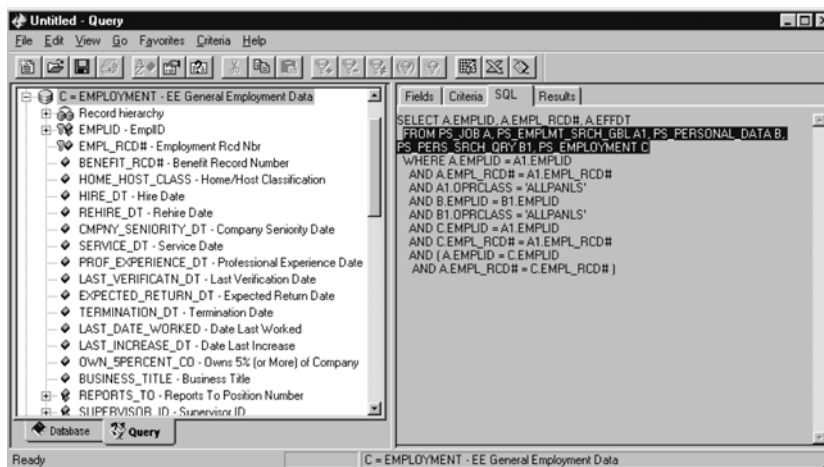


Figure 29.6 Adding the EMPLOYMENT table to our query

In addition to using the Query tool in PeopleSoft, data may also be selected from online panels. Let's take a closer look how PeopleSoft manages security in this case.

29.2.2 Row-Level security in online Panels

As an example, let's open the JOB_DATA panel group (figure 29.7).

Navigation: Go →PeopleTools →Application Designer →Open →Panel Group →JOB_DATA

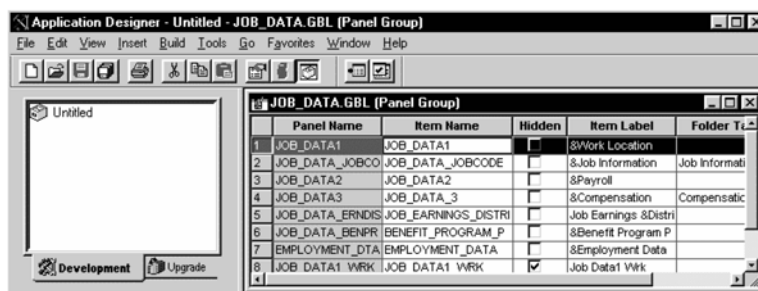


Figure 29.7 The JOB_DATA panel group

If you press ALT/ENTER, you can see the Properties Panel (figure 29.8).



Figure 29.8
The EMPLMT_SRCH_US search record is used for the JOB_DATA panel group

As you can see from figure 29.8, the EMPLMT_SRCH_US search record is used for the JOB_DATA panel group. What that means is that this view is responsible for bringing up the Selection dialog box with all rows that correspond to both the user's criteria and the security criteria. For example, if the user specified a partial name search as "Smith," not all rows from the database with the name starting with "Smith" will be returned, only the ones to which the user has access based on the department security setup. In chapter 24 of this book, we ran a trace to learn exactly what PeopleSoft is doing behind the scenes when the search criteria has been entered. As we saw in chapter 24 when the user specifies the search criteria, PeopleSoft builds the SQL statements to select from the Access Search record specified for a particular panel group.

29.3 PREVENTING AN SQR PROGRAM FROM EXECUTING OUTSIDE THE PROCESS SCHEDULER

As we discussed, in online PeopleSoft processing, data selection is controlled by PeopleSoft Security. This is not necessarily true when you execute your batch processes. You know that SQR programs in PeopleSoft can be submitted in several different ways—from the Process Scheduler, from the SQRW dialog box, or from the command line.

If you're running your SQR programs through the Process Scheduler, their execution is controlled by levels of PeopleSoft Security. First, you must be an authorized PeopleSoft user to login to the PeopleSoft system. Second, menu security is in place to prevent unauthorized access to a particular menu or panel. Third, when creating a process definition, you must specify authorized process security groups, thus

Process Scheduler. If the value of `$prcs_oprid` is `Null`, SQR displays an error message and exits the program.

This simple code can be easily implemented in any custom program, thus ensuring program execution only from the PeopleSoft Process Scheduler.

TIP If you need to bypass this security check during your testing stage you can enclose the bolded code in the `#ifdef/#endif` statements.

29.4 INCORPORATING ROW-LEVEL SECURITY IN SQR

In release 7.5, PeopleSoft incorporated the Row-Level security for its HRMS application. This implementation consists of both the online and batch modifications. The online changes are performed by using PeopleTools, and once implemented, they will apply to all programs in the HRMS application. The batch modifications are done on a program-to-program basis.

Take a look at this feature in the `INSTALLATION` table.

Open the Third Party panel from the Installation Table panel group (figure 29.9).

Navigation: Go → Define Business Rules → Define General Options → Setup
→ Installation Table

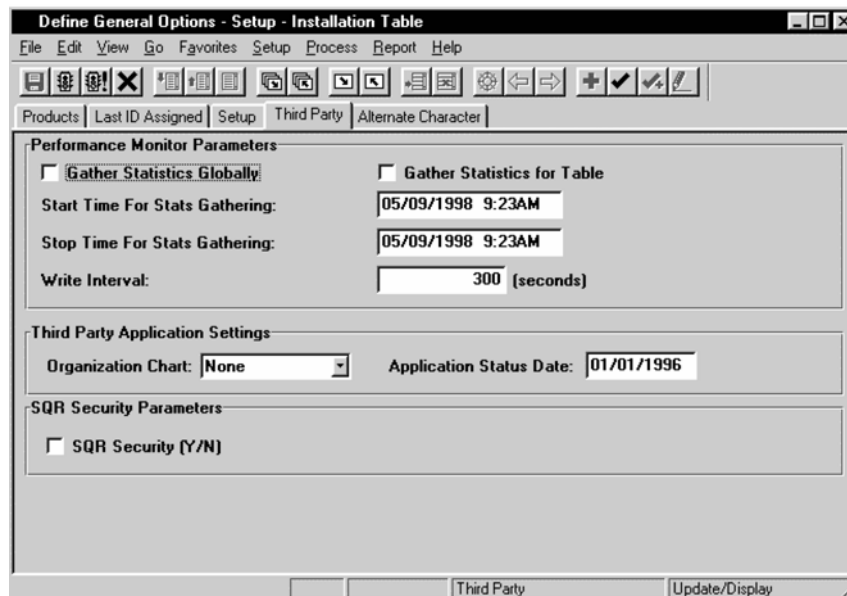


Figure 29.9 SQR Security flag is added to the Installation Table

As you can see from figure 29.9, the SQR Security flag is added to the Third Party panel. This is currently delivered for the PeopleSoft HRMS application. If you want to activate SQR Security, you should set the flag ON. If flag is not on, the system will not be using SQR Security features.

WARNING In order to use the SQR Security delivered by PeopleSoft you should also implement the PeopleSoft's Fast Security to populate the security views that are used in the delivered SQR programs.

Let's see now how SQR Security is implemented in PER003.sqr

Listing 29.1

PER003.sqr

```
begin-report
  do Init-DateTime
  do Init-Number
  Move 1 to $Year4
  do Init-Report
  if $prcs_oprid=''
    goto last2
  end-if
  if $scrty_flag='Y'
    do Process-Main-Scrty
  else
    do Process-Main
  end-if
  do Reset
  do Stdapi-Term
last2:
end-report

...

begin-procedure Init-Report

  move 'PER003' to $ReportID
  do Delete-Worktable
  do Stdapi-Init
  if $prcs_oprid=''
    display ''
    display 'REPORT CAN NOT BE EXECUTED OUTSIDE OF PEOPLESOFT, PLEASE USE
PROCESS SCHEDULER.'
    display ''
    goto last1
  end-if
  do Sqr-Param
```

If Security is Activated, execute
Process-Main-Scrty, otherwise
execute Process-Main

The Sqr-Param procedure
selects Security flag value
from the Installation table.

```

if $prcs_process_instance = ''
  do Ask-As-Of-Date
  do Ask-Years-Of-Service
else
  do Select-Parameters
end-if
do Init_Printer
do Init_Report_Translation ($ReportID, $language_cd)
do Append_Report_Translation ('HR')
last1:
end-procedure

...
begin-procedure Process-Main

  move '1' to $Year4
  move '-' to $DDelimiter
  do Format-DateTime($AsOfDate, $AsOf_YMD, {DEFYMD}, '', '')
  do Data-Selection
  do Create-Report
  do Delete-Worktable

end-procedure

begin-procedure Process-Main-Scrty

  move '1' to $Year4
  move '-' to $DDelimiter
  do Format-DateTime($AsOfDate, $AsOf_YMD, {DEFYMD}, '', '')
  do Data-Selection-Scrty
  do Create-Report
  do Delete-Worktable

end-procedure
...
#include 'hrsecty.sqc'    !Get SQR Security parameters

```

**HRSECTY.sqc
contains the
SQR-Param
procedure**

As you can see in the above excerpt from PER003.sqr program, the program calls the SQR-Param procedure at the beginning. This procedure is located in the HRSECTY.sqc file. Its only job is to select the Security flag from the INSTALLATION table. Based on the selection, the program either uses the security features or ignores them. That's why, when PeopleSoft modified the old version of this program to incorporate the security features, it duplicated the data selection procedure leaving intact the old ones to be able to run the program in two modes—with and without security. Take a look at the Process-Main-Scrty procedure. It's a clone of the Process-Main, the only difference being that it calls another data selection procedure: Data-Selection-Scrty.

Let's compare two routines—Data-Selection and Data-Selection-Scrty—and see how PeopleSoft incorporated the row-level security.

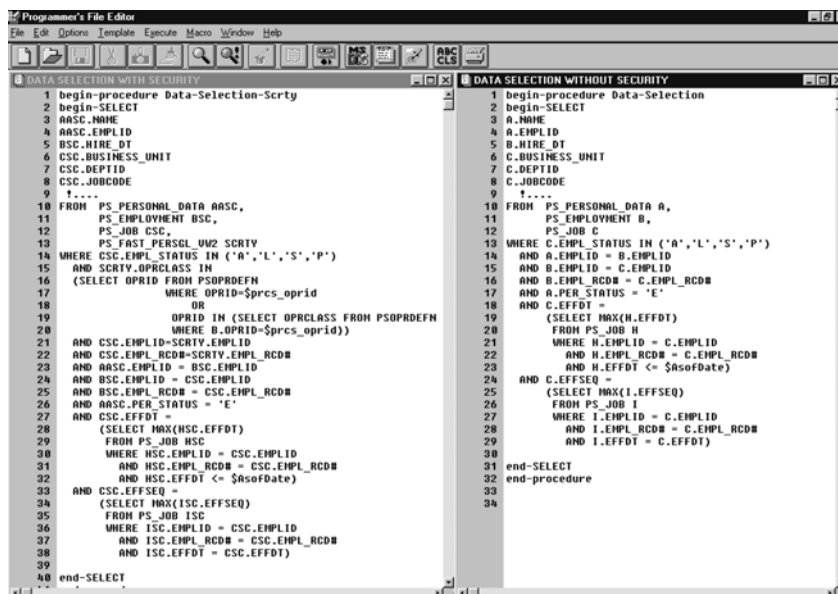


Figure 29.10 Data-Selection and Data-Selection-Scrtcy

Take a closer look at the two procedures displayed in figure 29.10. The one on the left is the new data selection with security, while the other on the right, is the procedure without security. For visibility purposes and to simplify the comparison, all irrelevant SQR statements between the last selected column and the FROM keyword were deleted from both procedures since they were absolutely identical.

The procedure with security has all the functionality of the one without security, plus more. Take a look at line number 13 on the left side. You can see that a new table, PS_FAST_PERSGL_VW2, is added to the SQL join. This is a fast department security view that plays the same role as the Query Security view that we discussed earlier in this chapter. This view is joined with the PS_JOB table by EMPLID and EMPL_RCD#. In addition (see line number 15) OPRCLASS, which is a key field in the security view, is matched with the \$prcs_oprid. In other words, the Select statement only selects the records of the employees that the operator (\$prcs_oprid) is authorized to access.

The fast security view PS_FAST_PERSGL_VW2 is an alternate fast search record for PS_PERS_SRCH_GBL. In order to use this view, the Fast Security delivered by PeopleSoft must be implemented. Fast Security uses the Application Engine to populate a special security table that was created to support search views with faster performance. Note that you do not have to use fast security search records in order to implement security in your SQR programs. Instead of the PS_FAST_PERSGL_VW2 record, the PS_PERS_SRCH_QRY search record could be used. If you have already implemented Fast Security, you can take advantage of both batch and online fast security access.

Let's summarize now what we have learned about implementing security in SQR and what practical steps have to be undertaken to support this security.

!!! Practical steps necessary to implement security in SQR for HRMS applications:

- make certain that SQR Security flag is checked ON in Installation table
- include the hrsecty.sqc file in your SQR program
- add code to your program to make sure your SQR program is executed from the Process Scheduler by verifying the `$prcs_oprid` variable
- call the SQR-Param procedure to select the SQR security flag from the Installation table
- check if SQR security Flag is On or Off and call the corresponding procedure (with or without security)
- create an additional data selection procedure that will be called if the SQR security flag is ON. Add a security search record to this procedure to restrict access to non-authorized data

TIP When OPRCLASS is retrieved based on OPRID in a sub-select statement (figure 29.10, lines 16-20) the SQL performance is impacted negatively, because the sub-select is executed for every selected row. To improve performance, obtain OPRCLASS only once outside of the main select procedure. Then use the retrieved OPRCLASS in the main select.

Now, that we know how simple it is to implement security in SQR, can we go ahead and incorporate it in all our custom programs? The process may not always be that straight-forward. You need to know your data and choose a correct search record for your data selection procedure. If your goal is to limit the employee selection to authorized operators only based on the department security implemented in your system, you can use either the fast security record or query security records. Both these views utilize the department security. If your SQR program selects neither employees nor departments, you need to come up with some other alternatives. In the next section, we will show how you can limit selection at the Run Control panel level without changing your SQR program.

29.5 USING RUN CONTROL RECORDS FOR SQR SECURITY

When executing SQR programs from the Process Scheduler, users are often asked to enter parameters online via a Run Control panel. These parameters are saved in the corresponding Run Control records and then passed to your SQR. Of course, not all SQR programs accept input parameters. We will consider a case when parameters are entered from an online Run Control panel and will show you how you can implement a simple security system by using PeopleTools.

Let's consider, for example, a typical Payroll task—running a Paysheets report.

Figure 29.11 shows a standard Run Control panel used in most Payroll batch processes. In order to run a process (in this case it is the Paysheets report), the operator has to either specify a pay run ID or enter the three parameters on the right of the panel: Company, Pay Group, and Pay End Date. Let's use the second method and enter the parameters into the panel. Click on the Company field and you can see the list of all available companies (figure 29.12). When Company is selected, click on Pay Group, and you can see all the paygroups for the selected Company (figure 29.13).

Navigation: Go →Compensate Employees →Manage Payroll Process →Report 2 →Paysheets

Figure 29.11 A Typical Run Control Panel for a Payroll Process

Co	Descr
CJP	Continental Commerce - Japan
CLA	Continental Commerce - Latin Am
CMX	Continental Commerce - Mexico
CND	Continental Commerce - Nethln
CUK	Continental Commerce - U.K.
HCT	Allied Health Care Services
LCI	Local State/County Government
MQ4	Midwest Manufacturing
MDB	Multi Dimensional Business
NCS	New Century Services
PGW	Power, Gas & Water
PST	Payroll Services Technology
RTH	RTH, Inc.
R15	R15, Inc.
R15	R15, Inc.

Figure 29.12 The Company Prompt

As you can see, an operator can select any Company/Paygroup combination available from the prompt. What if we make Company and Paygroup selection dependent on operator class? This will facilitate implementation of the online Company/Paygroup security.

- 1 Create a Company/Paygroup Security table to control data access for operators (operator classes) based on Company/Paygroup combinations.

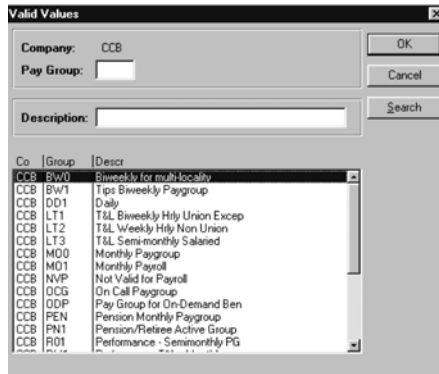


Figure 29.13 The Paygroup Prompt

- 2 Create a panel for online control of Company/Paygroup Security.
- 3 Create the company, paygroup, and run ID views to be used as online prompt tables for Company and Paygroup in all records related to any Payroll Process run. This will limit the process execution to operator classes that have access to authorized Company/Paygroups.
- 4 Modify the Run Control record definition to use the new prompt records.

Our objective is to demonstrate how the online security can be implemented. Following are the highlights of the Company/Paygroup security implementation.

Step 1

Creating Company/Paygroup Security table, MY_COMP_PAYGRP (figure 29.14).

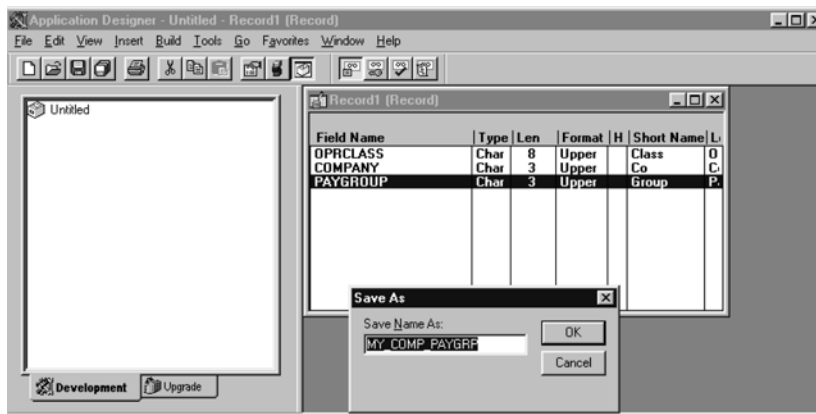


Figure 29.14 Creating Company/Paygroup Security table

Step 2

Creating Company/Paygroup Security Maintenance panel (figure 29.15).

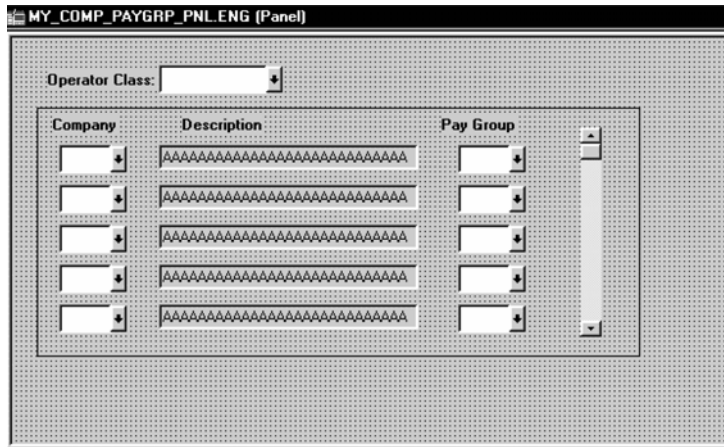


Figure 29.15 Company/Paygroup Security panel

After creating a panel group and adding it to the setup menu, we can populate the Company/Paygroup Security panel with the appropriate information. For each operator class, users must enter all combinations of Company and Paygroups a particular operator class can access. Figure 29.16 shows how the maintenance of the Company/Paygroup Security is performed.

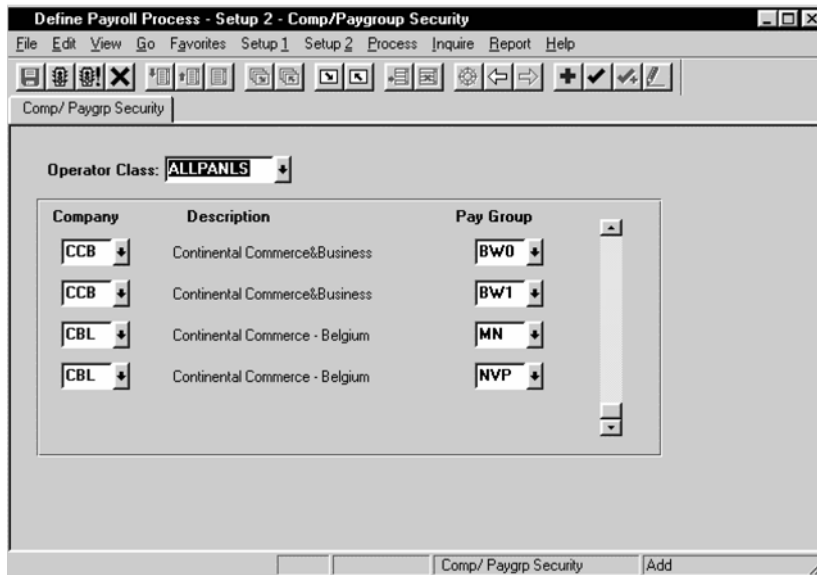


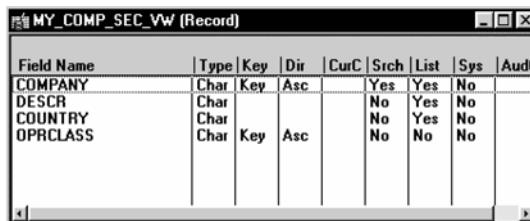
Figure 29.16 Assigning companies and paygroups to the ALLPANLS operator class

Let's get back to our development. We need to create the necessary views for our prompt records.

Step 3

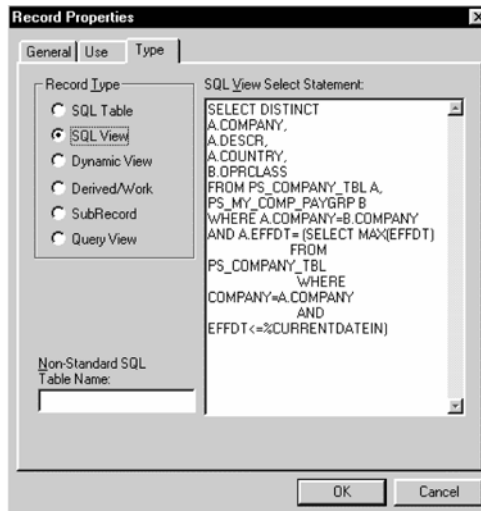
Creating the company, paygroup, and run ID views.

We need to create three views that may be used as prompt tables for Company, Paygroup, and Run ID respectively. Since we already created a security table (MY_COMP_PAYGRP) that contains all Company/Paygroup combinations per operator class, the view creation is simple. Figures 29.17 thru 29.22 show each of the views with their definitions.



Field Name	Type	Key	Dir	CurC	SrcH	List	Sys	Audt
COMPANY	Char	Key	Asc		Yes	Yes	No	
DESCR	Char				No	Yes	No	
COUNTRY	Char				No	Yes	No	
OPRCLASS	Char	Key	Asc		No	No	No	

Figure 29.17
The Company security view



Record Properties

General | Use | Type

Record Type:

- ☐ SQL Table
- ☒ SQL View
- ☐ Dynamic View
- ☐ Derived/Work
- ☐ SubRecord
- ☐ Query View

Non-Standard SQL Table Name:

SQL View Select Statement:

```
SELECT DISTINCT
A.COMPANY,
A.DESCR,
A.COUNTRY,
B.OPRCLASS
FROM PS_COMPANY_TBL A,
PS_MY_COMP_PAYGRP B
WHERE A.COMPANY=B.COMPANY
AND A.EFFDT=(SELECT MAX(EFFDT)
FROM
PS_COMPANY_TBL
WHERE
COMPANY=A.COMPANY
AND
EFFDT <= %CURRENTDATEIN)
```

OK Cancel

Figure 29.18
The SQL statement for the
Company security view

Field Name	Type	Key	Dir	CurC	Srch	List	Sys	Audt
COMPANY	Char	Key	Asc		Yes	Yes	No	
PAYGROUP	Char	Key	Asc		Yes	Yes	No	
OPRCLASS	Char	Key	Asc		No	No	No	

Figure 29.19
The Paygroup Security view

Record Properties

General Use Type

Record Type:

- ☐ SQL Table
- ☒ SQL View
- ☐ Dynamic View
- ☐ Derived/Work
- ☐ SubRecord
- ☐ Query View

SQL View Select Statement:

```
SELECT
COMPANY,
PAYGROUP,
OPRCLASS
FROM PS_MY_COMP_PAYGRP
```

Non-Standard SQL Table Name:

OK Cancel

Figure 29.20
The SQL statement for the Paygroup Security view

Field Name	Type	Key	Dir	CurC	Srch	List	Sys	Audt	H
RUN_ID	Char	Key	Asc		Yes	Yes	No		
OPRCLASS	Char	Key	Asc		No	No	No		
DESCR	Char				No	Yes	No		

Figure 29.21
The Run ID Security view

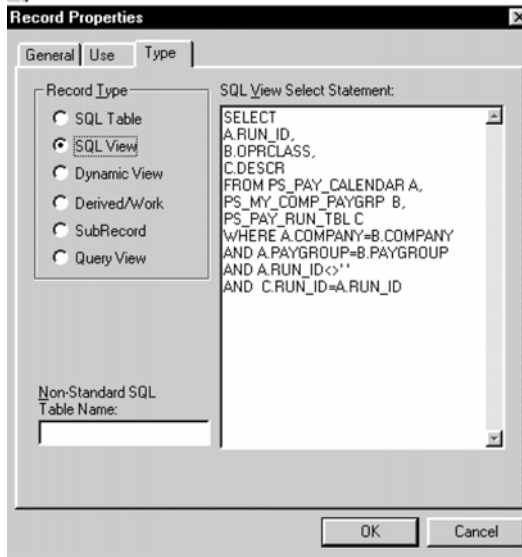


Figure 29.22
The SQL statement for the
run ID security view

Our views have been created. We should, of course, build them on a database level. After all is done, we can move on to the next step.

Step 4

Modifying Run Control record definitions.

In the first three steps, we prepared a basis for our security. Now, we need to replace the current prompt tables with the new views we just created. Since we already discussed the Paysheets report, let's find the Run Control record used for this report, then modify this record.

Take a look at the Order panel for the RUNCTL_PAYINIT2 panel used in the Paysheets report (figure 29.23).

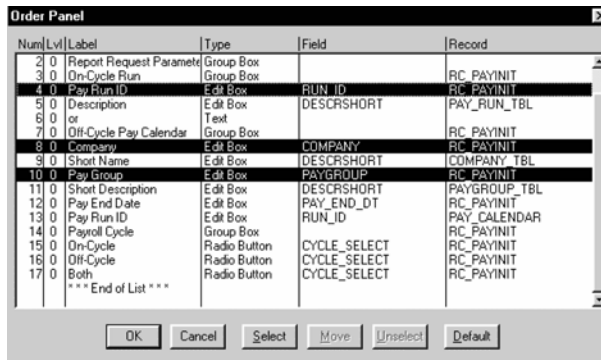


Figure 29.23
The RC_PAYINIT Run Control
record used for the
paysheets report

To implement this customization we have two alternatives:

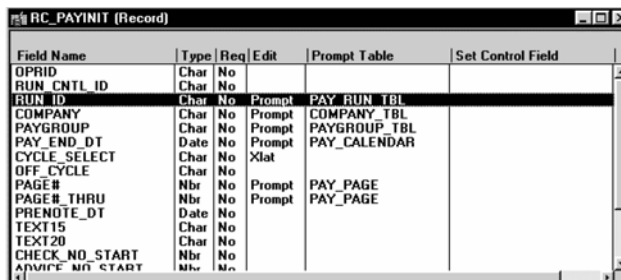
- Customize the existing RC_PAYINIT record.
- Save this record as MY_RC_PAYINIT, then customize it. In addition, all panels that are linked to the RC_PAYINIT record and the corresponding panel groups must be cloned as well.

Generally, we do not recommend modifying PeopleSoft-delivered records, but our modifications in this case are not structural and, therefore, do not require the corresponding database level alterations. All panels that use this record with no modifications will benefit from the Company/Paygroup based security. During the upgrade process, the Compare and Report process will recognize the difference in the RC_PAYINIT record.

The second alternative requires more objects to be modified.

Therefore, selecting the first alternative, our customization of the RC_PAYINIT record is limited to replacing the prompt tables with our custom views.

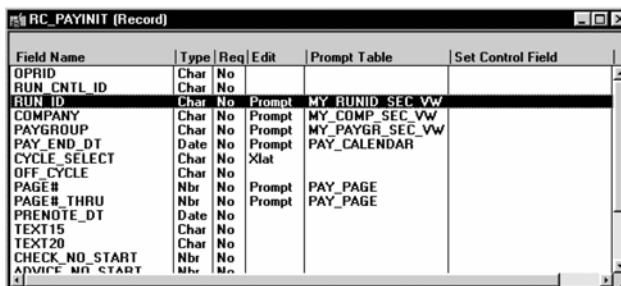
Let's open the record and modify it by replacing the values for prompt tables in Run_ID, Company, and Paygroup fields with the newly created prompt tables (figure 29.24).



Field Name	Type	Req	Edit	Prompt Table	Set Control Field
OPRID	Char	No			
RUN_CNTL_ID	Char	No			
RUN_ID	Char	No	Prompt	PAY_RUN_TBL	
COMPANY	Char	No	Prompt	COMPANY_TBL	
PAYGROUP	Char	No	Prompt	PAYGROUP_TBL	
PAY_END_DT	Date	No	Prompt	PAY_CALENDAR	
CYCLE_SELECT	Char	No	Xlat		
OFF_CYCLE	Char	No			
PAGE#	Nbr	No	Prompt	PAY_PAGE	
PAGE#_THRU	Nbr	No	Prompt	PAY_PAGE	
PRENOTE_DT	Date	No			
TEXT15	Char	No			
TEXT20	Char	No			
CHECK_NO_START	Nbr	No			
ADVISE_MN_START	Nbr	No			

Figure 29.24
The Prompt tables that are currently used in RC_PAYINIT record

After replacing the prompt tables for Company, Paygroup, and Run_ID, our table looks like that in figure 29.25.



Field Name	Type	Req	Edit	Prompt Table	Set Control Field
OPRID	Char	No			
RUN_CNTL_ID	Char	No			
RUN_ID	Char	No	Prompt	MY_RUNID_SEC_VW	
COMPANY	Char	No	Prompt	MY_COMP_SEC_VW	
PAYGROUP	Char	No	Prompt	MY_PAYGR_SEC_VW	
PAY_END_DT	Date	No	Prompt	PAY_CALENDAR	
CYCLE_SELECT	Char	No	Xlat		
OFF_CYCLE	Char	No			
PAGE#	Nbr	No	Prompt	PAY_PAGE	
PAGE#_THRU	Nbr	No	Prompt	PAY_PAGE	
PRENOTE_DT	Date	No			
TEXT15	Char	No			
TEXT20	Char	No			
CHECK_NO_START	Nbr	No			
ADVISE_MN_START	Nbr	No			

Figure 29.25
The RC_PAYINIT record with the new prompt tables

The only step remaining is to test everything together. As you may remember, after the Company/Paygroup security table and maintenance panel were created, we just associated the ALLPANLS operator class with four combinations of two companies and two paygroups for each company (figure 29.16). This means that, if we try to select the company, paygroup, or Run Control ID values, from the modified Run Control record, we should only be able to select these values. Let's give this a try (figure 29.26).

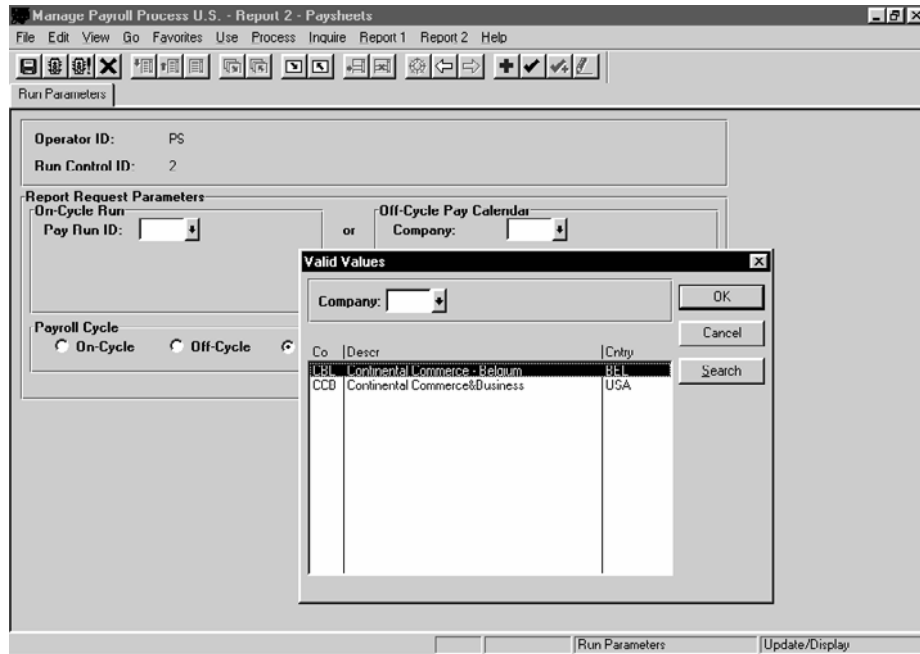


Figure 29.26 The prompt in Company field only shows two companies to which the ALLPANLS has access to

After selecting the Company value, let's select the Paygroup (figure 29.27).

We can also verify if our run ID prompt view works. Let's select another Run Control and, this time, enter the values on the left portion of the panel (figure 29.28).

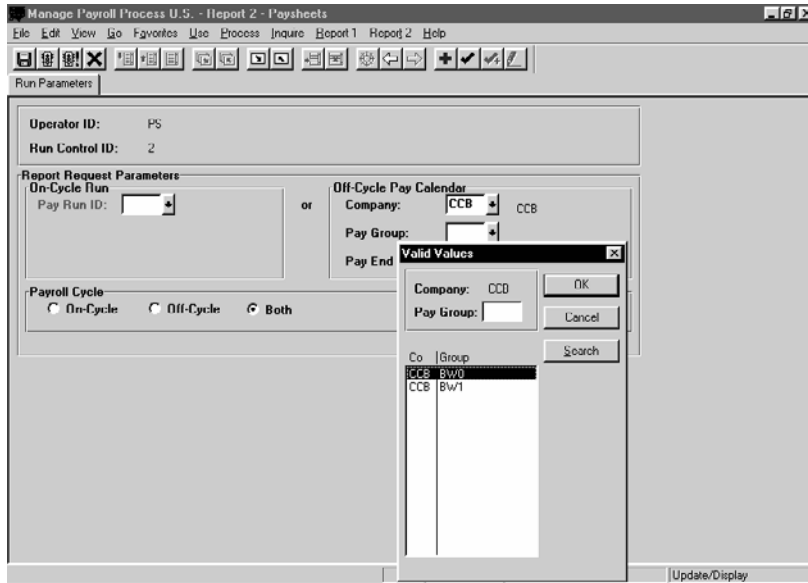


Figure 29.27 The prompt in the Paygroup field only allows to select from two paygroups

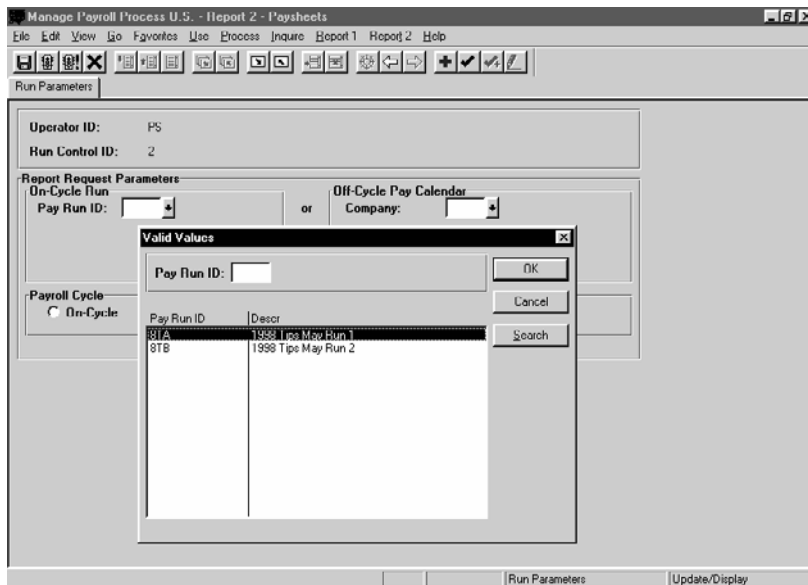


Figure 29.28 The prompt in the Run ID field only shows the run IDs that are valid for the company/paygroup to which we have access

As you can see, all our modifications are working. With this simple customization we can control user security via an online panel without changing your programs. We demonstrated this change for the Paysheets process, but in fact, any program that uses the modified Run Control record RC_PAYINIT will now be secured from unauthorized execution. If your report uses other Run Control records, you need to modify the prompt records accordingly.

We based the Company/Paygroup security on the combination of these two fields but a similar approach can be used to implement security based on other key fields.

As you can see from our examples in this chapter, you may need to make only a slight code modification in your SQR program to ensure that the program will run only under the Process Scheduler. We have demonstrated how you control your SQR security without altering your SQR program in any significant way.

KEY POINTS

- 1 PeopleSoft provides you with layers of online security to help protect your data from unauthorized access.
- 2 When accessing a PeopleSoft application in a networked environment, you have to pass through network security, database security, and PeopleSoft online security.
- 3 PeopleSoft delivers a special way of controlling online access to your data rows with the help of security search records.
- 4 In the Query tool, PeopleSoft automatically joins its tables with the corresponding security views.
- 5 In release 7.5, PeopleSoft incorporated the row-level security for its HRMS applications. This implementation consists of both the online and batch modifications. The batch modifications are done routinely on a program-to-program basis.



CHAPTER 30

Additional Process Scheduler topics

- 30.1 Scheduling programs for execution on a recurring basis 684
- 30.2 Using job streams 688

During the course of this book, we've discussed the execution of both PeopleSoft-delivered and custom processes with the help of the PeopleSoft Process Scheduler. We demonstrated how to create a process definition to execute SQL programs and how to monitor the process execution status. You also learned how to communicate with the Process Scheduler via API programs. However, there are some other important aspects of the PeopleSoft Process Scheduler that we did not cover.

30.1 SCHEDULING PROGRAMS FOR EXECUTION ON A RECURRING BASIS

When you execute your programs on the Server, you can schedule them to run at pre-defined intervals. A special recurrence definition has to be created and assigned to the process. When the recurrence definition is created, it may be assigned to the process through its process definition or from the Process Request Dialog panel at runtime.

Let's see how we can schedule our custom program, MYPROB02.sqr for execution every Sunday at 8:00 A.M.

Navigation: Go → Problem Tracking → Report → Status Report → Add



Figure 30.1
Adding Run Control ID for every Sunday 8 A.M. run

TIP Give each Run Control a meaningful name and use it for one process only.

The next screen is our Run Control panel. Since we are planning to schedule the Status Report for execution every Sunday, we have to supply the appropriate parameters to our SQR program every time the program runs. When the program is executed manually, users enter the parameters when they submit the process for execution. We need to find a way to automatically fill in our Run Control table. You can use several methods, depending on your business needs. The simplest method is to use the system date as the *As Of Date* parameter. The standard method is to make your program default to system date if the date in the Run Control record is blank. If you take a look at our MYGETVAL.sqc, you can see that this is exactly what we did. If the system date technique is not applicable to your process, you must develop your own custom method to automatically supply the correct date to your program at each program execution. The second parameter, *Problem Status*, can be entered only once, and the Run Control record retains its value for all the subsequent runs.

Suppose, we are going to use the system date as the *As Of Date* parameter and a blank to indicate that we need the status report of all the problem statuses. In this case, our Run Control panel is saved as shown in figure 30.2.

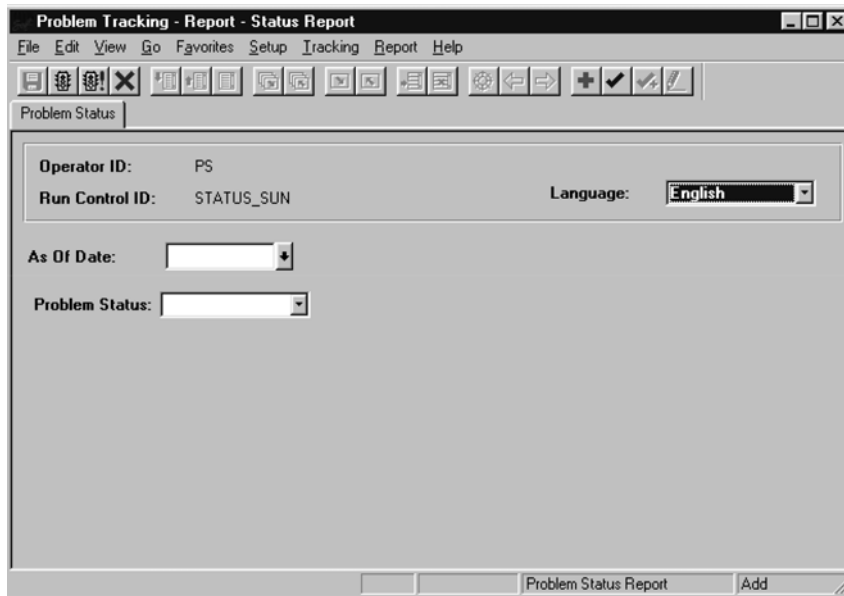


Figure 30.2 Leaving blank values in both parameters to use the system date as As of Date and to make the program report all problem statuses

Click on the Traffic Light to go to the Process Request panel, make sure that the process Run Location is Server, and start creating a new recurrence definition, by clicking on the New button in the Run Recurrence box.

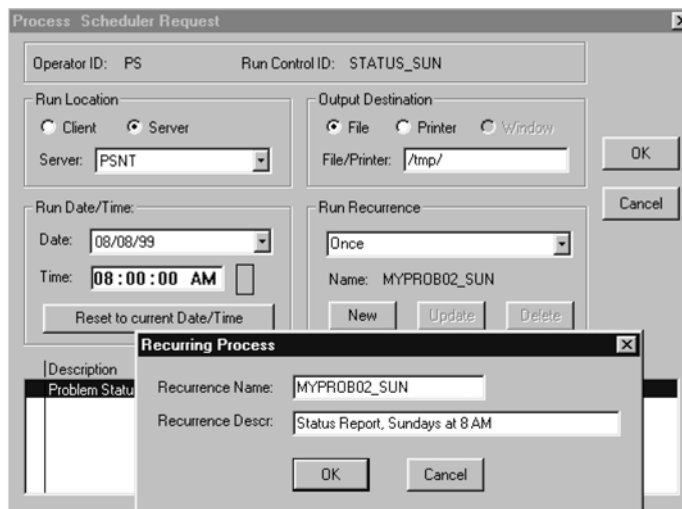


Figure 30.3
Creating a new Run
Recurrence

After clicking on the OK button, we need to specify our new recurrence information (figure 30.4).

Status Report, Sundays at 8 AM

Occurs

☐ Every Day ☐ Bi-Weekly

☐ Every Weekday ☐ Monthly

☒ Weekly ☐ Yearly

Start Request

On: 08/22/99

At: 08:00:00 AM

Reset to current Date/Time

Repeat

Every: For:

Start next recurrence when:

☒ Prior recurrence has completed.

☐ Next recurrence is scheduled.

Process will be scheduled

Every week on Sunday, starting 08/22/99 at 08:00:00 AM

OK

Cancel

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figure 30.4
Specifying the recurrence information

In the panel shown in figure 30.4, we clicked on Weekly to specify the occurrence frequency, then we selected the starting day and time of the cycle. We also clicked on Su (Sunday), and the system automatically selected all Sundays in the calendar. Based on our selections, the system created a meaningful description in the lower box of the panel. You should always verify this description to make sure that all schedule parameters are correct. We also selected the option to schedule the next run when the prior run has completed. This means that the next job will be queued only when the previous job is completed successfully.

Now we are ready to click on the OK button.

Operator ID: PS Run Control ID: STATUS_SUN

Run Location: ☐ Client ☒ Server
Server: PSNT

Output Destination: ☒ File ☐ Printer ☐ Window
File/Printer: /tmp/

Run Date/Time: Date: 08/22/99 Time: 08:00:00 AM
Reset to current Date/Time

Run Recurrence: Status Report, Sundays at 8 AM
Name: MYPROB02_SUN
New Update Delete

Description	Name	Process Type Descr
Problem Status Report	MYPROB02	SQR Report

OK Cancel

Figure 30.5 The MYPROB02 process is about to be scheduled

Note that, even though we are not planning to execute the process now, we must hit the OK button to schedule our process for recurrent executions. So far, we have created a process recurrence definition where we have specified all the parameters for our process scheduling, but we have not yet scheduled the process. We need to do it manually the first time in order to make the scheduling take place. After that, the Process Scheduler will do the job.

Let's click on the OK button again and verify the process status on the Process Monitor panel.

As you can see in figure 30.6, our process is queued and is scheduled for execution at 8 A.M. on 08/22/1999. As soon as this process has successfully executed, the next occurrence of this process will be scheduled automatically by the Process Scheduler in accordance with your process recurrence definition.

WARNING When scheduling another process for recurring processing, be very careful if you decide to reuse an existing recurrence definition. Changing any parameters in the recurrence definition may result in taking the first process out of the scheduler.

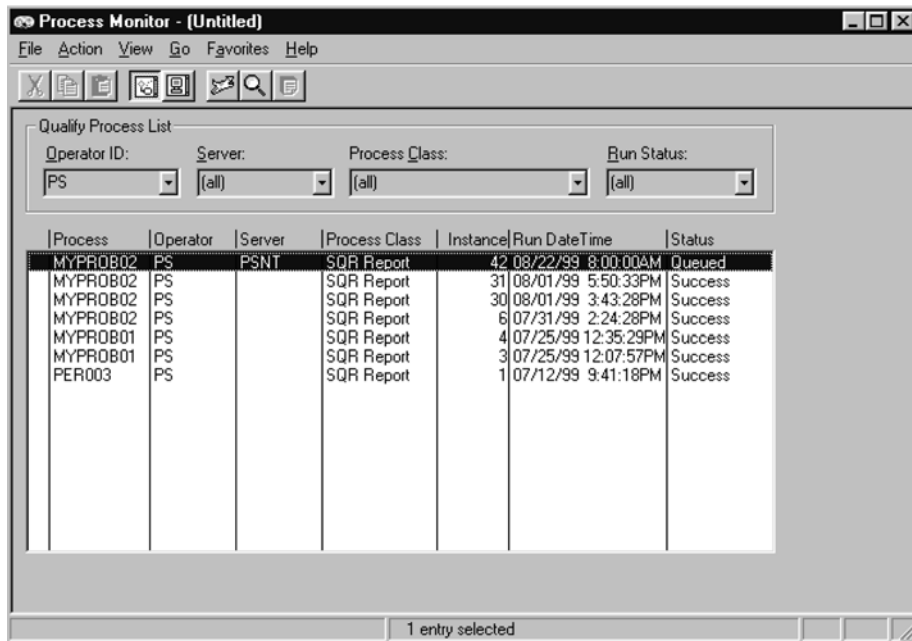


Figure 30.6 MYPROB02 is scheduled for execution

30.2 USING JOB STREAMS

So far we have been executing and scheduling single processes from the PeopleSoft Process Scheduler. Oftentimes, your business requires the execution of multiple processes one after another or in parallel. PeopleSoft allows you do this if you run your processes on Server. The job definition is used to accomplish this task.

A job (or job stream) in PeopleSoft usually consists of two or more processes. You can combine your SQR and COBOL programs into one Job to be executed in a parallel or serial mode. When scheduling your job to run in a serial mode, all processes within the job will be executed sequentially, one after another. Otherwise, they will be executed in a parallel mode without any specific order. As with an individual process, you can schedule a job to run at a later time or on a recurring basis. It is always a good approach to combine all processes, which should be executed at a specific time (for example, nightly), into a job stream and schedule this job stream to be executed at pre-defined time intervals (for example, every night at 10 PM).

In the following examples, we will create a job stream and schedule it for execution.

Exercise 1

Execute the Refresh Employees process (PER099.sqr) and the problem status report (MYPROB02.sqr) in a job stream every night at 10 P.M.

In order to schedule any job for execution, a job definition has to be created. Since our exercise calls for the execution of two reports, we need to create a job definition that contains PER099.sqr and MYPROB02.sqr. As we already know, when we execute a process from the Process Scheduler, this process accepts its input parameters from online panels. A process definition is linked to a specific panel through a panel group. Like a process definition, a job definition also requires a panel group to be specified. Therefore, all processes in your job stream accept input parameters from this particular panel group, which may consist of several panels. To illustrate this point, let's create a panel group for our job stream.

30.2.1 Creating a panel group for a job stream

Since the processes we are going to include in our job stream are already designed to run under the Process Scheduler, our task is simple. We just need to combine their Run Control panels into one panel group.

Let's find out what the components are for our new panel group. In order to do so, we take a look at the process definitions for PER099.sqr and MYPROB02.sqr (figure 30.7). We start with PER099.sqr.

Navigation: Go →Administer Workforce →Administer Workforce (U.S.) →Process →Refresh Employees Table →Update/Display

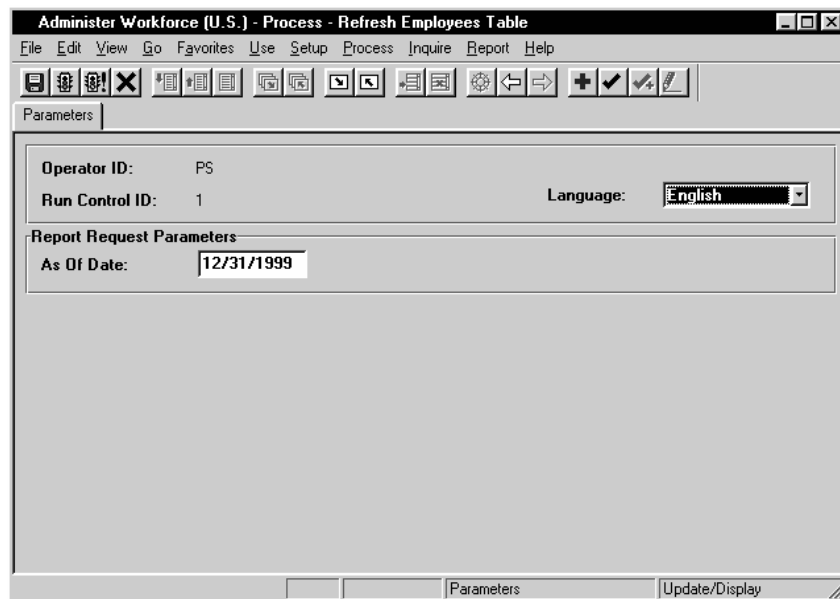


Figure 30.7 Finding the name of the Run Control panel for the Refresh Employees Table process

The Run Control panel used to run the Refresh Employees Table process is RUNCTL_ASOFDAT. When scheduling this program for execution in a job stream, we obviously want to preserve all the functionality of the job's components, including the input parameters processing. Therefore, we include this panel into our new panel group.

We know from the previous chapter that the name of the Run Control panel for our problem status report is MY_RUN_CNTL_PRB01.

Let's now create a new panel group.

Navigation: Go →PeopleTools →Application Designer →New →Panel Group

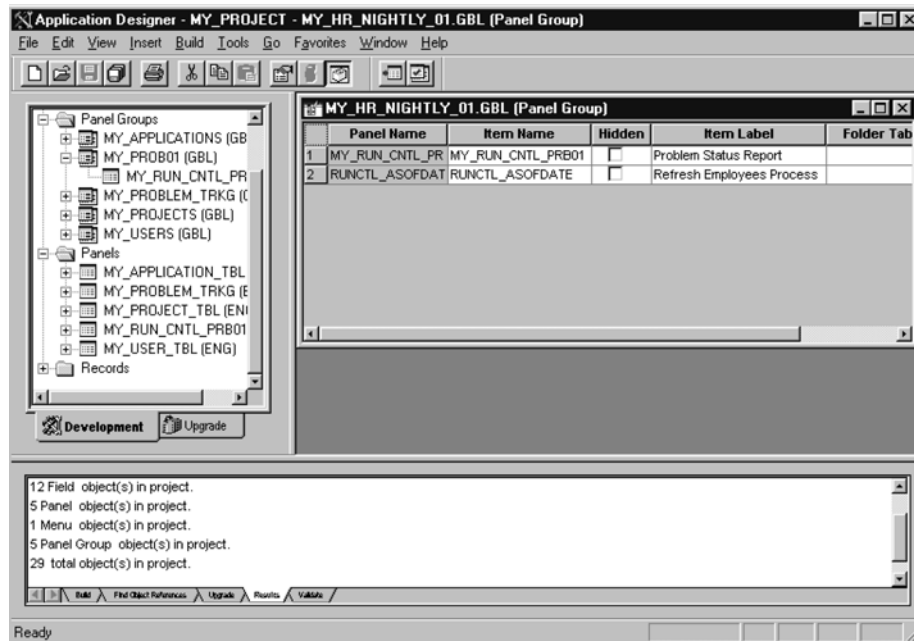


Figure 30.8 Creating a new panel group for a job stream

Our new panel group includes both the MY_RUN_CNTL_PRB01 and the RUNCTL_ASOFDAT Run Control panels. After putting meaningful labels for each panel in the panel group, we need to specify the Panel Group Properties (figure 30.9).

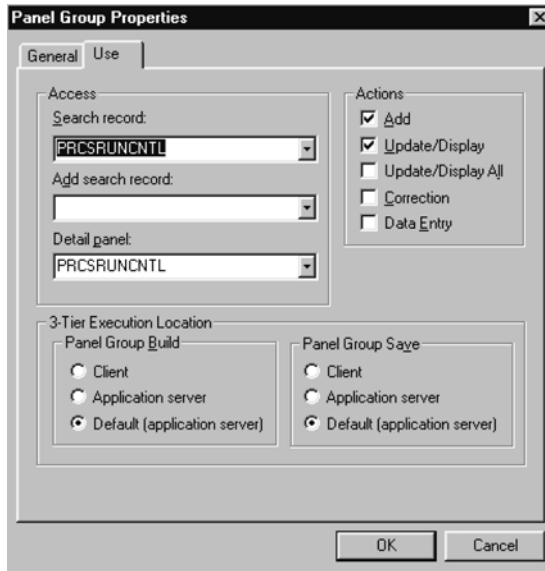


Figure 30.9
Specifying panel group properties
for our job stream



Figure 30.10 Saving panel
group for new job stream

The Search record and the Detail panel for our Run Control panel group should be the same as the one for a panel group in a single process.

Now, we save the new object as MY_HR_NIGHTLY_01 (figure 30.10).

After clicking on the OK button, we are ready to add our job to a menu.

30.2.2 Creating a Menu Item for our new job stream

As usual, in order for our users to access a Run Control panel, we need to attach this panel to an appropriate menu item via a panel group. Let's create a new menu bar, Job Stream, and use it for our new job stream menu item and for all future job streams.

As we can see in figure 30.11, we created a new menu bar, Job Stream. Then, just by clicking on an empty rectangle under this menu bar, we created a new menu item, HR Nightly, and linked our MY_HR_NIGHTLY_01 panel group to this menu item.

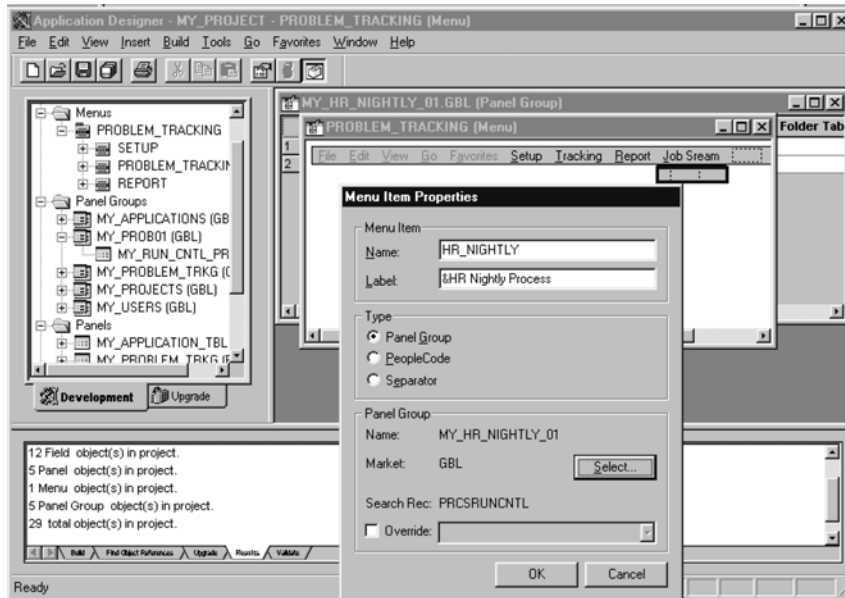


Figure 30.11 Creating a new menu bar and menu item

After clicking on the OK button, we can modify the Operator's Security to allow the ALLPNLS operator group access to our new menu item. Now, we can test the menu (figure 30.12).

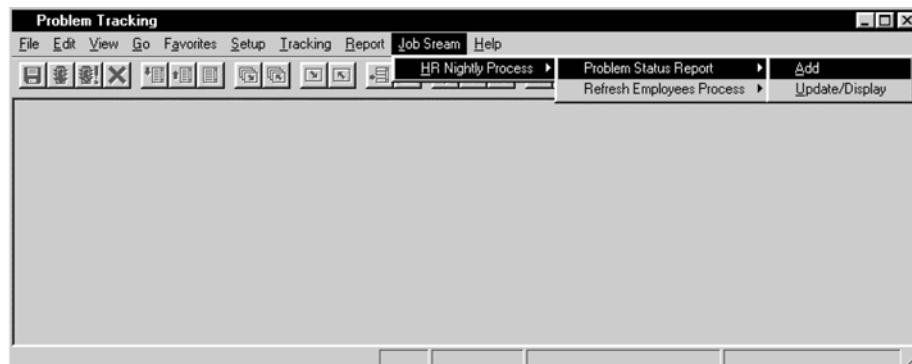


Figure 30.12 New menu bar and menu item are created for our job stream

Are we ready to run our job? Not yet. We need to create a job definition first.

30.2.3 Creating a job definition

Unlike a process definition creation, when adding a new job definition, the process job name does not have to match any of your processes. You can give any name to your job.

Navigation: Go →PeopleTools →Process Scheduler →Use →Job Definitions →Job Definitions →Add



Figure 30.13 Specifying a job definition name

The process of creating a job definition is similar to that of creating a process definition. Let's take a close look at what is involved in this process and discuss the meaning of each field in the job definition.



Figure 30.14 Creating a job definition

In the *Job Description*, you specify the job definition description that will be displayed on the Process Scheduler Request panel.

The *Server Name* should be specified only if you want to restrict your Job execution to a specific server. If you leave it blank, the system will find an available server based on the Process class.

You can specify a new *Process Class* by entering a unique process class name, or you can select it from a drop-down list. Usually if all the processes included in your job stream are SQR programs, you would select SQR Report as your process class.

TIP If you include both SQR programs and other programs such as COBOL, or programs written in Application Engine, you need to select Programs as the Process class.

In our case, since we are executing two SQR programs, we select SQR Report.

The *Job Run Mode* can be either *Serial* or *Parallel*. If you want your processes to be executed sequentially, you select *Serial* mode, otherwise, use *Parallel*. We will run our processes in a parallel mode since the second report does not depend on the first process execution.

The *Job Priority* could be *High*, *Medium*, or *Low*. This information is used by the Process Scheduler to initiate jobs with higher priorities first. We'll specify a *Medium* priority for our job.

The *Recurrence Name* is used to specify a recurrence schedule that you previously set up. This parameter is optional and can be defined for your JOB on the Process Scheduler Request panels.

In the lower portion of your job definition panel, specify the processes you want to include in the job. If you've selected a *Serial* mode, your processes have to be listed in the order in which they will be executed. In prior releases, you had to number each item sequentially with no gaps. Release 7.5 takes care of the numbering automatically. It also re-numbers the processes when you need to add a new item between existing ones or change the order of processes in your job.

You should turn on the *Run Always* flag if you want your processes to be executed, even if one of the previous processes failed. Suppose, for example, you selected the mode as *Serial* and did not turn on the Run Always flag for any of your processes. Let's assume that your job contains three processes: Process1, Process2, and Process3. If your Process2 fails, you will see the following process statuses in the Process Monitor screen (assuming that you clicked on the job's "+" sign to see the individual processes):

Process	Status
- MyJob	Error
Process1	Success
Process2	Error
Process3	Hold

After you check the error messages in the log file and fix the problem, you need to restart your JOB from Process2. Currently, PeopleSoft Process Scheduler does not have capability to restart the Job from a specific point. You could either re-execute the entire Job or execute Process2 and Process3 as individual processes. Be careful. It's not always safe to execute certain processes again because your processes may be updating the database.

In the second panel of the JOB Definitions panel group, you specify the Process Security Groups for the users to whom you would like to give permissions to run your job. You also specify the panel group to which your job should be attached. For our job, we specify the name of the panel group that we created to run this Job.

Our job definition is created. Let's save it and see how we can schedule our job to run recurrently.

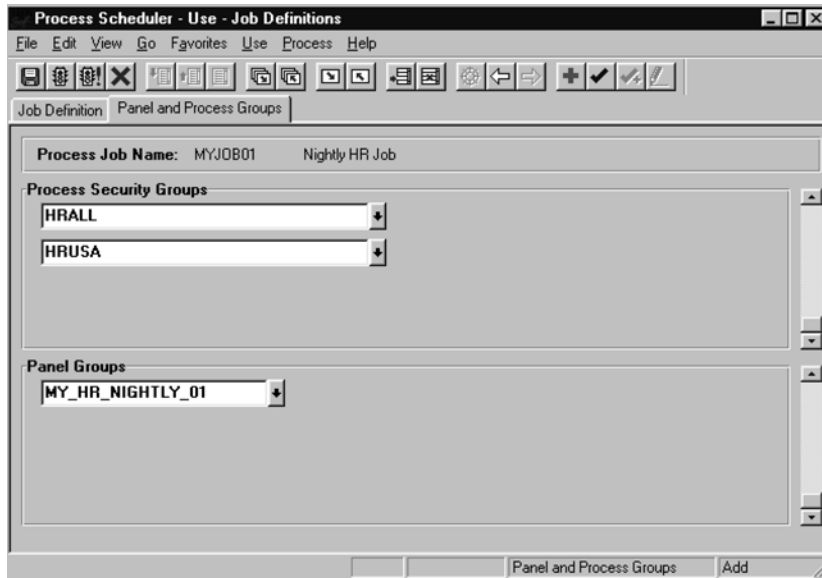


Figure 30.15 Specifying the Process Security Groups and the panel group for our job

30.2.4 Scheduling a job for recurrent execution

Navigation: Go → Problem Tracking → Job Stream → HR Nightly Process → Add



Figure 30.16 Adding a new Run Control record for job's execution

Let's add a new Run Control ID, myjob01, and click on the OK button. The Run Control panel group appears (figure 30.17).



Figure 30.17 Entering input parameters

As shown in figure 30.17, our panel group contains two panels. The first one is the Problem Status Report panel, and the second one is the Refresh Employees Process panel. Does this mean that, if you have ten processes in your job definition, you need ten panels in your Run Control panel group? Not necessarily. Some of your processes may not require any input parameters and, therefore, do not need additional panels. Others may need the same input parameters. In that case, one panel may be used for several processes. It all depends on the Run Control records which your processes use to get the input parameters.

TIP You need to make sure that all Run Control records used by the processes in your job are present in the panel group.

Let's take, for example, the Years of Service program. It accepts two parameters: *As Of Date* and *Years of Service*. If you have one job that includes this program and another one that only needs *As of Date* as its input parameter—for example, the Pending Future Actions report—you can use the same panel to run both reports. This is all, of course, under condition that both programs are using the same Run Control record.

In the second panel of our panel group, the Refresh Employees Process Run Control panel (figure 30.18), we also use the `As Of Date` input parameter, but since the Run Control records for both our processes are different, we have to include both panels in our panel group.

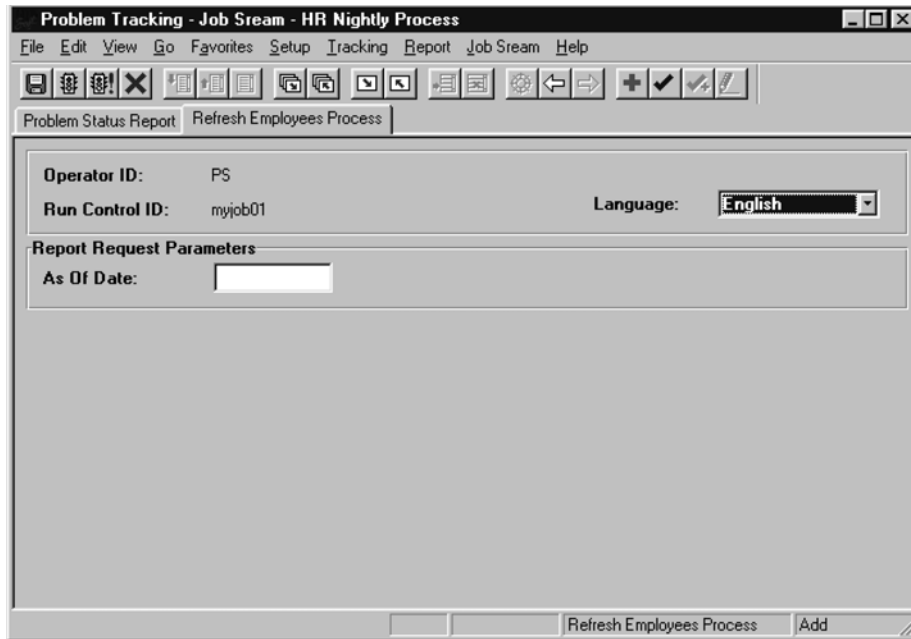


Figure 30.18 The Refresh Employees Run Control panel

Since we are planning to schedule our job to be executed on a recurrent basis, we leave the `As Of Date` field blank to force the process to use the system date instead. This way, we make our scheduling much simpler.

After all parameters in the Run Control panels have been entered, we are ready to schedule our job for execution. Let's click on the Traffic Light.

As you can see in figure 30.19, we specified the Run Location as `Server`, and we also selected a specific Server name from the Server drop-down list. Please note that jobs can only be scheduled on `Server`. On the bottom of the panel, you can see our job name displayed. If you click on the plus sign to the left of the job, as we did, both the processes that make up the job will be shown. When scheduling the job for execution, select the Nightly HR Job (figure 30.19).

Process Scheduler Request

Operator ID: PS Run Control ID: myjob01

Run Location
☐ Client ☒ Server
 Server: PSNT

Output Destination
☒ File ☐ Printer ☐ Window
 File/Printer: /tmp/

Run Date/Time
 Date: 08/21/99
 Time: 02:19:00 PM
 Reset to current Date/Time

Run Recurrence
 Once
 Name:
 New Update Delete

Description	Name	Process Type Descr
Nightly HR Job	MYJOB01	PSJob
Refresh Employees Table	PER099	
Problem Status Report	MYPROB02	

Figure 30.19 Process Scheduler Request panel for MYJOB01 job stream

Let's now create a new Run Recurrence based on the user's request to run this job daily at 10 P.M. To do so, click on the New button in the Run Recurrence group box.

Figure 30.20 shows the parameters we set for our new recurrence definition. Let's click on the OK button and give our recurrence definition a meaningful name.

Nightly HR Job at 10 PM

Occurs
☒ Every Day ☐ Bi-Weekly
☐ Every Weekday ☐ Monthly
☐ Weekly ☐ Yearly

Start Request
 On: 08/16/99
 At: 10:00:00 PM
 Reset to current Date/Time

Repeat Every: [] [] For: [] []

Start next recurrence when:
☒ Prior recurrence has completed.
☐ Next recurrence is scheduled.

Process will be scheduled
 Every day starting 08/16/99 at 10:00:00 PM

OK Cancel

Figure 30.20
Creating a new definition

Process Scheduler Request

Operator ID: PS Run Control ID: myjob01

Run Location:
☐ Client ☒ Server
 Server: PSNT

Output Destination:
☒ File ☐ Printer ☐ Window
 File/Printer: /tmp/

Run Date/Time:
 Date: 08/16/99
 Time: 10:00:00 PM
 [Reset to current Date/Time]

Run Recurrence:
 [Nightly HR Job at 10 PM]
 Name: MYJOB01
 [New] [Update] [Delete]

[OK] [Cancel]

Description	Name	Process Type Descr
Nightly HR Job	MYJOB01	PSJob
Refresh Employees Table	PER099	
Problem Status Report	MYPROB02	

Figure 30.21 Process Scheduler Request with Run Recurrence

Our job is ready for execution. We click on the OK button to schedule the job to run for the first time (figure 30.22).

Process Monitor - (Untitled)

File Action View Go Favorites Help

Qualify Process List

Operator ID: PS Server: (all) Process Class: (all) Run Status: (all)

Process	Operator	Server	Process Class	Instance	Run Date/Time	Status
MYJOB01	PS	PSNT	SQR Report	38	08/16/99 10:00:00PM	Queued
PER099	PS			1	08/16/99 10:00:00PM	Queued
MYPROB02	PS			2	08/16/99 10:00:00PM	Queued
MYPROB02	PS		SQR Report	31	08/01/99 5:50:33PM	Success
MYPROB02	PS		SQR Report	30	08/01/99 3:43:28PM	Success
MYPROB02	PS		SQR Report	6	07/31/99 2:24:28PM	Success
MYPROB01	PS		SQR Report	4	07/25/99 12:35:29PM	Success
MYPROB01	PS		SQR Report	3	07/25/99 12:07:57PM	Success
PER003	PS		SQR Report	1	07/12/99 9:41:18PM	Success

1 entry selected

Figure 30.22 MYJOB01 is scheduled for execution on 08/16/99 at 10 P.M.

As you can see from figure 30.22, our job MYJOB01 includes two processes, PER099 and MYPROB02. Its status is *Queued*, and it is scheduled for execution on 08/16/99 at 10 P.M. As soon as the job is executed successfully, the system will automatically schedule MYJOB01 for execution on the next day, 08/17/99 at 10 P.M.

KEY POINTS

- 1** You can schedule programs that run on the Server for execution on a recurring schedule.
- 2** A job (or job stream) may include more than one process.
- 3** In order to schedule a job for execution, a job definition has to be created. Similar to a process definition, a job definition also needs to be associated with a panel group.
- 4** You can include processes of different types (SQR program, COBOL, Application Engine) into one job.
- 5** Make sure that all Run Control records used by the processes in your job are present in the panel group.
- 6** If you want processes to be executed sequentially, you should select the *Serial* mode on your job definition panel; otherwise, routinely use the *Parallel* mode.



CHAPTER 31

SQR and Process Scheduler—PeopleSoft 8

31.1 Process Scheduler terminology	703	31.5 Process Scheduler security	709
31.2 Process Definitions	703	31.6 Process Scheduler PeopleCode support	710
31.3 Process Scheduler Request dialog	706	31.7 SQR and PeopleTools 8	710
31.4 Output options	707		

PeopleSoft introduced a number of new and enhanced features in PeopleTools 8. These features improve the way in which SQR programs and other processes interact with the PeopleSoft Process Scheduler. PeopleSoft delivered enhancements in the following areas:

- Process Scheduler terminology
- process definitions
- Process Scheduler Request Dialog
- output options
- Process Scheduler security
- recurrence definitions
- Process Scheduler PeopleCode support

Let's take a quick tour of these modifications in PeopleTools release 8.

31.1 **PROCESS SCHEDULER TERMINOLOGY**

PeopleSoft modified its terminology to avoid confusion and simplify the Process Scheduler usage for application developers, system administrators, and end-users. The Process Scheduler tool contains the following components:

- Process Scheduler Manager
- Process Scheduler Request Dialog
- Process Request Monitor
- Process Scheduler Server Agent

Each of these components has a specific task and, depending on your role in the PeopleSoft world, you may or may not work with all the tools. Nonetheless, the knowledge of when to use a particular tool is essential. Therefore, the new names that PeopleSoft introduced are extremely helpful.

Table 31.1 Modified Process Scheduler terminology

Release 7.5	Release 8
Process Scheduler	Process Scheduler Manager
Process Monitor	Process Request Monitor
Process Types	Process Type Definitions
Process Servers	Server Definitions
Process System	System Settings

As you can see from table 31.1, the Process Scheduler has been renamed to the Process Scheduler Manager. You use the Process Scheduler Manager to create and maintain process types and process and job definitions. You access it by selecting Go →PeopleTools →Process Scheduler Manager.

Similarly, the Process Monitor has been renamed to the Process Request Monitor. It is accessed by selecting Go →PeopleTools →Process Request Monitor.

The menu items, Process Types, Process Servers, and Process System, have been renamed to Process Type Definitions, Server Definitions, and System Settings, respectively. You can find these menu items under the Process Scheduler Manager →Use menu.

31.2 **PROCESS DEFINITIONS**

The Process Definitions panel group now consists of four panels. The existing panels have been redesigned as well.

Do you remember that the Process name you enter in this add box should be your SQR name without the .sqr extension? Have you ever tried to enter a name of a non-existing SQR program? Before release 8 such a blunder was possible. You could easily create a process definition for a program that was never written. Starting from release 8, PeopleSoft only allows you to specify programs that can be found in the

Navigation: Go →PeopleTools →Process Scheduler Manager →Use →
Process Definition →Add



Figure 31.1 Adding a Process Definition

Configuration Manager’s SQRW search path parameter. This is a great step toward making the system more secure.

As you can see in figure 31.2, the Process Definition panel group has an additional tab, Override Options.

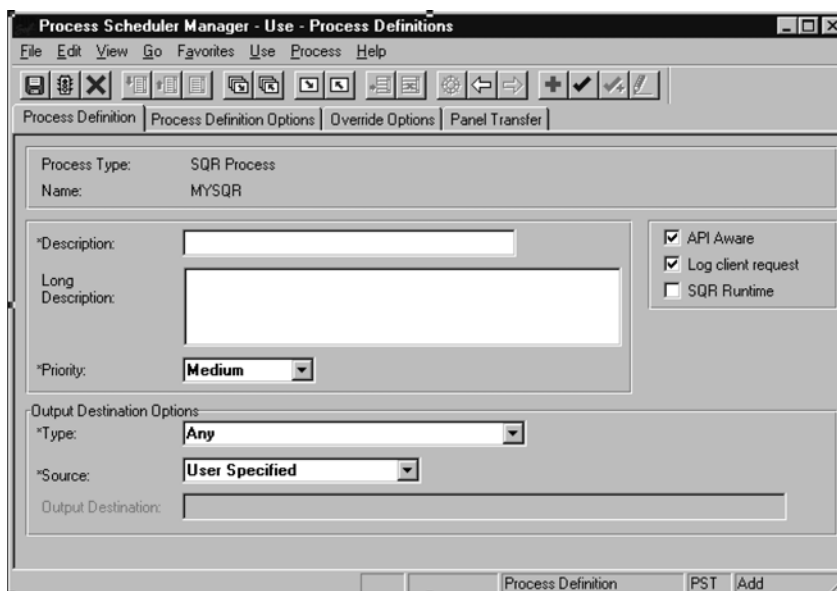


Figure 31.2 The Process Definition panel group in the Process Scheduler Manager menu

Let’s take a closer look at the Process Definition panel. If you compare this panel to that in release 7.5, you will notice that the panel, too, has changed. The Output Destination options has been moved from the Process Definitions Options panel to

this panel. The types of output available for selection are Printer, Window, Email, File, or Any. The option Any allows a user to specify any valid option.

The Source should still be selected as User Specified for SQR programs.

The Output Destination is only available when the Source is selected as the process definition, which means that this request will default to the output destination specified by the process definition.

Let's switch to the Override Options tab (figure 31.3).

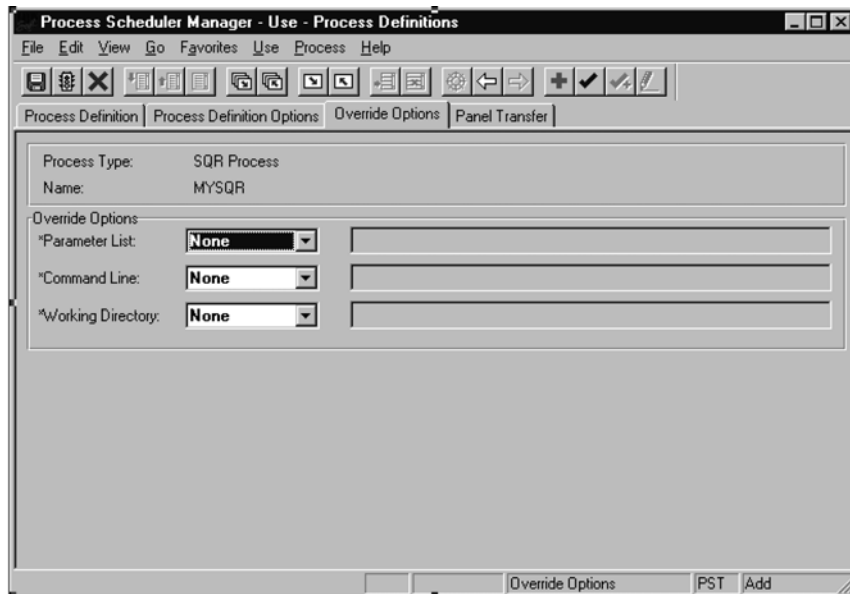


Figure 31.3 The Override Options tab

This panel is actually a simplified version of the old Process Definition Options. It allows you to specify the Override Options for the Parameter List, Command Line, and Working Directory.

How many times have you had your Process Scheduler definitions created incorrectly? Now, with release 8 tools, you will be able to run the SysAudit process to display incorrect Process Scheduler definitions.

31.3 PROCESS SCHEDULER REQUEST DIALOG

The Process Scheduler Request dialog box has been modified to simplify the end-user's run requests. Let's examine the panel in figure 31.4.

Description	Process Name	Process Type	Output Type	Output Format	Output Destination
COBOL Multi-process Job	3CBL	PSJob	File	(None)	% % Output Directory % %
Crystal Multi-process Job	3CRYSTAL	PSJob	File	(None)	% % Output Directory % %
DB Agent Multi-process Job	3DBAGENT	PSJob	File	(None)	% % Output Directory % %
SQR Multi-process Job	3SQR	PSJob	File	(None)	% % Output Directory % %
Application Engine Test	AETESTPROG	Application Engine	(None)	(None)	
All Process Types	ALLTYPES	PSJob	File	(None)	% % Output Directory % %

Figure 31.4 The Process Scheduler Request dialog in release 8.0

What makes release 8 exciting is that with it you are able to specify a different Output Type, Output Format, and Destination at the individual process level (including for each process within a job).

Release 8 also brought in some preventive measures that help identify problems before a process is submitted. For example, the Process type “PSJob” is disabled if you have `Client` set as the run location. Available output types and formats depend on particular process types. We’ll discuss new output types later in this chapter.

Notice that even though the Run Recurrence box is still in the new panel, the end-user will not be able to update the recurrence definition from the Process Scheduler Request dialog panel. The end-user can select the necessary run recurrence from the list of recurrences previously set up. In release 8, Recurrence definitions are created through the Process Scheduler Manager → Use menu item. The new Recurrence Definition panel is shown in figure 31.5.

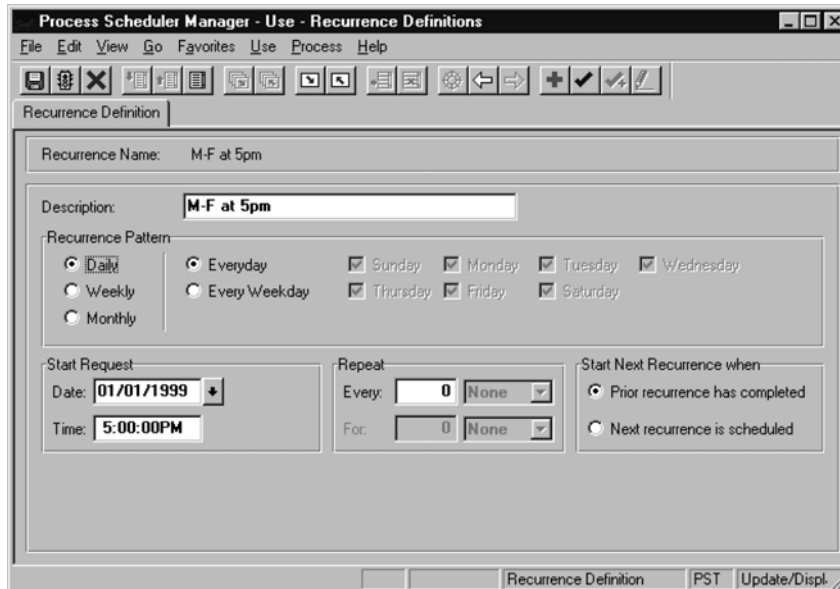


Figure 31.5 The Recurrence Definition panel in release 8.0

31.4 OUTPUT OPTIONS

Let's get back to figure 31.4. As you can see in the lower portion of the Process Request dialog panel, you can now specify the Output Type, the Output Format, and the Output Destination for each process. PeopleSoft introduced new output types and formats in release 8. To avoid any confusion in terminology, let's describe these three output options.

31.4.1 Output types

The output type tells the PeopleSoft Process Scheduler where the output of your process should go.

The following table shows the output types available in release 8.

Table 31.2 Output types

Output Type	Description	Available on Client	Available on Server
Window	Directs the process output to a DOS window.	Yes	No
File	Allows you to write the output to the file specified in Output Destination	Yes	Yes
Printer	Sends the output to the specified printer	Yes	Yes
Email	Sends the output to the predefined email list	No	Yes

As you can see from table 31.2, the long-awaited email file output is delivered as a new output type. You will be able to specify email addresses through the Security Administrator and send your output via email for applications executed on the Server.

31.4.2 Output formats

In addition to the output types, you will be able to select an appropriate Output Format, depending on what process type you have selected. If, for example, you select the process type as SQR Process, you have the following output format options:

- Acrobat(.pdf)
- Comma Delimited (.csv)
- HP format (.lis)
- HTML documents format (.htm)
- Line Printer Format (.lis)
- Postscript (.lis)
- SQR Portable Format (.spf)
- Other (.lis)

For *Crystal Reports* Process type the following formats will be available for your selection:

- Crystal Report (.rpt)
- HTML Document (.htm)
- Lotus 1-2-3 files (.wks)
- Microsoft Excel (.xls)
- Rich Text File (.rft)
- Text Files (.txt)

31.4.3 Output Destination

You can select an appropriate Output Destination depending on the Output Type and whether you run your process on Client or Server.

Table 31.3 shows the available options:

Table 31.3 Output Destination options

Output Type	Description	Client Output Destination	Server Output Destination
File	Directory output	Default Value: %OutputDirectory% defined in the Configuration Manager	Default Value: %%OutputDirectory%% defined in PSADMIN
Printer	Default printer	Printer defined for a Workstation	Printer defined for a Server
Email	Sends the output to a predefined email list	No	Email address

You can also enter the custom values for your file or printer output destinations if you have the proper security access.

In release 8, you will be able to select a printer from a list of installed printers in the Process Request Dialog panel.

You can also preset your process output type and output destination in the process definition. The values you specify will be reflected in the Process Request Dialog panel.

31.5 PROCESS SCHEDULER SECURITY

With Tools 8, the Security Administrator tool is also enhanced and allows you to access and set up the Process Profiles and Process Groups via a separate tab (figure 31.6).

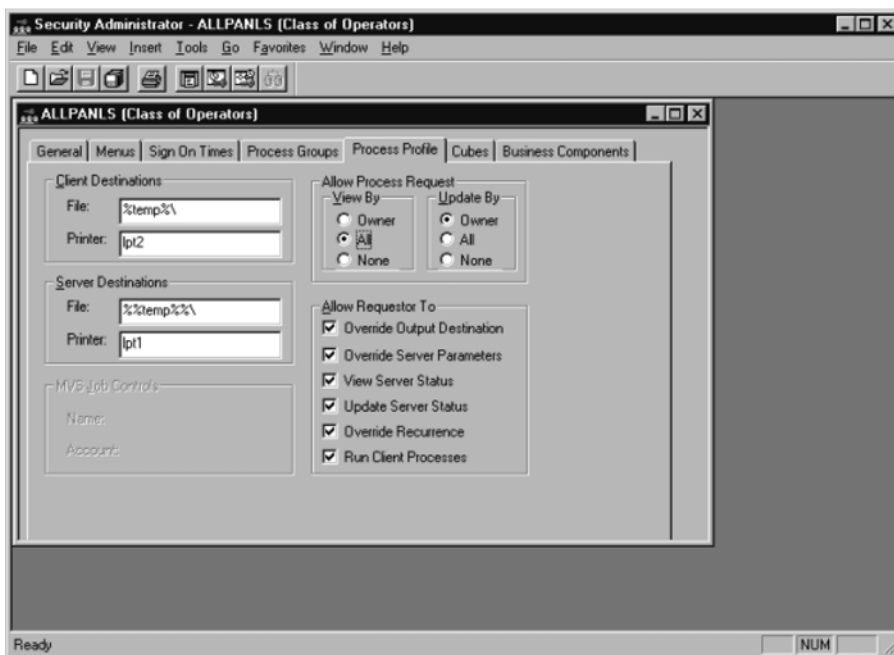


Figure 31.6 Security Administrator, Process Profile tab in release 8

Pay attention to the bottom checkbox in the Allow Requester To group box. By popular demand, PeopleSoft added an option to restrict certain classes of operators from running batch processes on their workstation.

In addition, the following changes were made to improve security in scheduling and executing processes:

- allowing the changing of an Operator/Access password after a process request is scheduled to run

- creating a “Super User” to monitor process requests via the Process Monitor
- offering an option to restrict users from scheduling recurring processes

31.6 PROCESS SCHEDULER PEOPLECODE SUPPORT

If you want to schedule your process from a PeopleCode script, PeopleSoft introduces a new Process Request PeopleCode Class. The goal is to make scheduling processes from PeopleCode easier. The `ScheduleProcess()` PeopleCode function is still supported in release 8, but it will be phased out in future releases.

The `ProcessRequest` class is used in release 8 for invoking processes through the Process Scheduler using PeopleCode. You can design a `ProcessRequest` PeopleCode program that can be triggered from a push button, a Save panel, or a field change event.

Using this new feature, you can schedule processes or jobs for immediate execution or in the future. It also supports the scheduling of recurring processes and jobs to run automatically at user-defined intervals.

31.7 SQR AND PEOPLETOOLS 8

As SQRiBe/Brio continues to improve its SQR application, PeopleSoft also integrates new SQR features to work smoothly with PeopleTools.

We already discussed in this chapter how PeopleSoft 8 improved the Process Request Dialog panel to incorporate Output Types, Output Formats, and Output Destinations. The SQR Output Formats now also include the PDF and CSV formats.

In addition, PeopleSoft 8 now supports the multiple report outputs.

31.7.1 Unique names for file output and logs

PeopleSoft 8 improved the naming of the output files and the log files. If a report is executed from the Process Scheduler the filenames will be:

< SQR Program Name>_<Instance>.xxx.

For example, if we executed the MYPROB01.sqr from the Process Scheduler and the Process Instance is 33876, the output filename will be MYPROB_33876.lis, and the log file name, MYPROB01_33876.log.

If the process instance is not available, the filenames are

<SQR Program Name>_<timestamp>.xxx

31.7.2 PSSQR shell

PeopleTools 8 delivers a new shell or SQR wrapper: the `PSSQR` executable. The `PSSQR` is an ANSI-C function consistent across different platforms. It replaces the script file that was previously used on Unix. This shell allows PeopleTools 8 to support output to HTML and PDF formats.

`PSSQR` also improves the delivery of reports to printers by employing a different technique of dealing with file outputs.

31.7.3 New printer setup SQCs

Since SQR output is processed differently in release 8, PeopleSoft developed new printer-independent versions of SETUPxx.sqc files. In order to use the new file output formats, you need to use either PTSET01.sqc or PTSET02.sqc files. The PTSET01.sqc was developed to replace PTPSP160.sqc, SETUP31.sqc, and SETUP01.sqc (whichever was used). The PTSET02.sqc is a replacement for PTPSL177.sqc, SETUP32.sqc, and SETUP02.sqc.

31.7.4 Additional features

PeopleTools 8 enhanced its support for the Global Time Zone and launched support of the Unicode.

Files will continue to be the primary way of integration. Release 8 brings in a simpler way of working with files. The File layout is now an object. It supports a graphical description of files. New robust file support features are added to PeopleCode as well. All this makes it easy for a third party to manipulate and exchange file layouts.

Understanding PeopleSoft COBOL

Many of PeopleSoft's major business processes are written in COBOL. PeopleSoft documentation usually includes a section called "Before You Customize," which cautions against making any modifications to the delivered COBOL processes. This is generally good advice. There are many issues to consider, including development cost, version upgrades, and continued PeopleSoft support. Regardless of whether or not one decides to customize, it's good practice to understand how the COBOL applications work. Undoubtedly situations will arise when an analysis of the COBOL programs will be required. The purpose of this section is to provide the reader with a basic understanding of PeopleSoft's COBOL techniques and how they are used to access the database. All database activity is processed through the use of a called module named PTPSQLRT. Because no direct SQL execution exists (outside of the PTPSQLRT module), the program structure and approach is consistent across all database platforms. This is one of the key ingredients to PeopleSoft's success as a provider of packaged solutions. As we discover how PeopleSoft COBOL is used, we can apply what we've learned by making a sample customization to a delivered application. Additional topics covered include the Process Scheduler API, Configuration Manager, and trace files.



CHAPTER 32

What's the difference?

32.1 Conventional COBOL programming	715
32.2 PeopleSoft structured programming	717
32.3 The PTPSQLRT module	720
32.4 Parameter descriptions	721
32.5 Setup lists	725
32.6 Action requirements	728

32.1 CONVENTIONAL COBOL PROGRAMMING

COBOL is often used in the client/server world. In fact, it is used just about everywhere! Mainframes, Unix systems, personal computers—all can utilize applications written in COBOL. COBOL is a portable programming language with little variance in the “core” language itself. Theoretically, the same program can be compiled successfully on different platforms with varying elements that can be utilized depending upon the platform you’re using. For example, some COBOL compilers have built-in functions (for running on a client-workstation) for keyboard handling, screen I/O (cursor control), and so forth. The main difference between conventional COBOL programming and PeopleSoft COBOL lies in the manner used in accessing the database. Instead of directly embedding your SQL statements, PeopleSoft uses a highly structured approach. All database access is controlled by calls to a delivered module called PTPSQLRT. SQL statements are stored in a database table and executed by passing the statement name and associated parameters to the PTPSQLRT module. We are going to explore how PeopleSoft uses this module to perform a variety of

functions, including connecting to the database, selecting, updating, inserting, and deleting records; and disconnecting from the database.

Before we dive into PeopleSoft's method of database access, let's take a quick look at how we access the database using embedded SQL in a conventional COBOL program.

32.1.1 Using SQL in COBOL programs

Many COBOL compilers allow you to embed SQL within the program. The following example uses Pro*COBOL. The SQL directives are identified by the EXEC SQL and END-EXEC commands, and the SQL communication area and working storage section are defined:

```
*      SQL Communications Area
      EXEC SQL INCLUDE SQLCA
      END-EXEC.
*****
*****  Declare Host Variables                      *****
*****
      EXEC SQL BEGIN DECLARE SECTION
      END-EXEC.
      ...
01  WS-SQL-WORK-AREAS.
    05  WS-USERID-PASSWD.
        10  USERNAME                      PIC X(10).
        10  PASSWD                        PIC X(10).
    05  EMPLID-LAST-EMPL                  PIC S9(08) COMP.
      ...
      EXEC SQL INCLUDE PSTABLES
      END-EXEC.
      ...

      ...
      EXEC SQL END DECLARE SECTION
      END-EXEC.
```

In the sample below we set a default error-handling routine. When any SQL error is encountered, the procedure Z999-SQL-ERROR is performed:

```
EXEC SQL WHENEVER SQLERROR
      DO PERFORM Z999-SQL-ERROR
      END-EXEC.
```

It is a good practice to prompt the user for a user ID and password instead of hard-coding them. Once the user ID and password are entered, a CONNECT command is executed:

```
DISPLAY 'Enter Username: ' WITH NO ADVANCING.
ACCEPT USERNAME.
DISPLAY 'Password       : ' WITH NO ADVANCING.
ACCEPT PASSWD WITH NO-ECHO.
```

```
EXEC SQL CONNECT
      :USERNAME IDENTIFIED BY :PASSWD
END-EXEC.
```

Following we select the column EMPLID_LAST_EMPL from the table PS_INSTALLATION and place the data into the bind variable EMPLID-LAST-EMPL, defined in the data division of the program:

```
EXEC SQL SELECT  EMPLID_LAST_EMPL
              INTO :EMPLID-LAST-EMPL
              FROM PS_INSTALLATION
END-EXEC.
```

The table PS_INSTALLATION is updated using the bind variable EMPLID-LAST-EMPL to populate the column EMPLID_LAST_EMPL:

```
EXEC SQL UPDATE PS_INSTALLATION
              SET  EMPLID_LAST_EMPL = :EMPLID-LAST-EMPL
END-EXEC.
```

Any updates are then committed to the database:

```
EXEC SQL COMMIT WORK
              RELEASE
END-EXEC.
```

As you can see, all SQL access is controlled by the programmer. The structure is free form and can vary by developer. PeopleSoft uses a far different approach, one that may seem cumbersome at first. Once you discover the secret to PeopleSoft's methodology, however, you'll find it much easier to analyze and, if necessary, modify PeopleSoft-delivered COBOL processes.

32.2 **PEOPLESOFT STRUCTURED PROGRAMMING**

All PeopleSoft COBOL programs use the same structured approach. There is no direct database access using embedded SQL. All SQL statements to be executed must reside in a database table called PS_SQLSTMT_TBL. PeopleSoft COBOL modules requiring database access through SQL are accompanied by a Data Mover script that contains all the SQL statements used by the module. The SQL statement table is populated by running the Data Mover script. This is how the SQL statements are initially loaded. Any modifications or additions to stored SQL statements should be made to the script and loaded using Data Mover. The key behind PeopleSoft's structured approach is the use of a main database activity module called PTPSQLRT. This module ensures consistency from one PeopleSoft program to another regardless of database platform or operating system.

To demonstrate the functionality of the PTPSQLRT module, let's use one of PeopleSoft's less complex processes—the process to delete obsolete monthly payroll balances (called PSPDLBAL)—as an example:

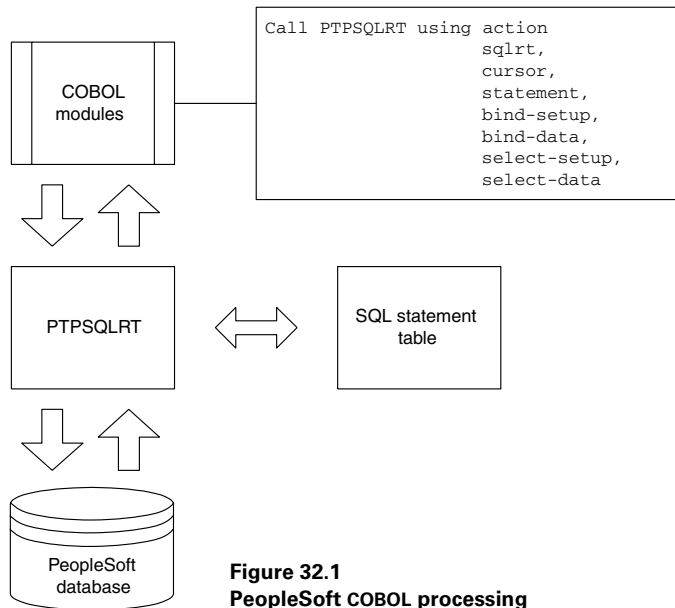


Figure 32.1
PeopleSoft COBOL processing

Figure 32.1 illustrates the overall design of PeopleSoft COBOL processing. Calls to PTPSQLRT are used to perform all database access functions. Stored SQL statements are retrieved from the SQL statement table and processed.

32.2.1 Stored SQL statements

First, let's take a quick look at the stored SQL statement table.

SQLSTMT_TBL	Stored SQL Statement Table
PGM_NAME	Program Name
STMT_TYPE	Statement Type
STMT_NAME	Statement Name
STMT_TEXT	Statement Text

Each COBOL program that calls a stored SQL statement has at least one entry. In our upcoming example, we'll access SQL statements with a PGM_NAME of PSPDLBAL. The statements are also qualified by a STMT_TYPE, which designates the type of SQL statement. The valid types are S (Select), U (Update), I (Insert), and

D (Delete). Finally, the STMT_NAME field is used to assign a unique statement name. The actual statement text is stored in the STMT_TEXT column. This statement text will be retrieved and compiled by PTPSQLRT.

32.2.2 Storing SQL statements from Data Mover scripts

The PSPDLBAL COBOL process comes with a Data Mover script (PSPDLBAL.DMS) which is loaded into the table PS_SQLSTMT_TBL. Let's look at a portion of this script:

```
STORE PSPDLBAL_S_RUNCTL
SELECT COMPANY,
        BALANCE_ID,
        BALANCE_YEAR,
        BALANCE_PERIOD
FROM PS_PAY_DBAL_RUNCTL
WHERE OPRID          = :1
      AND RUN_CNTL_ID = :2
;
```

Our first line contains a Data Mover STORE command. This is used to store the subsequent text in the STMT_TEXT column of the stored SQL statement table. The STORE command parameter PSPDLBAL_S_RUNCTL is used to identify the key elements. The parameter is a consolidated form of the SQL statement table keys. They are then broken down by Data Mover into the PGM_NAME, STMT_TYPE, and STMT_NAME columns of the SQL statement table. When we discuss the PTPSQLRT module in detail, you'll discover the SQL statements are accessed using the same consolidated method of identifying the statement.

The SQL statement text following the STORE command is placed in the SQL statement table. Notice the use of bind variables :1 and :2. All bind variables are entered in this manner. When the statement is retrieved, the bind variable is resolved and the statement processed.

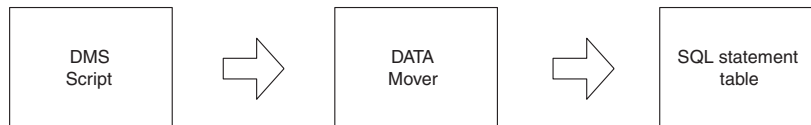


Figure 32.2 Data Mover processes DMS script (stores SQL statements)

Figure 32.2 depicts the Data Mover process of loading stored SQL statements. The DMS script contains all the stored SQL statements. Data Mover loads the SQL statements into the SQL statement table where they can then be utilized by the COBOL processes through calls to PTPSQLRT.

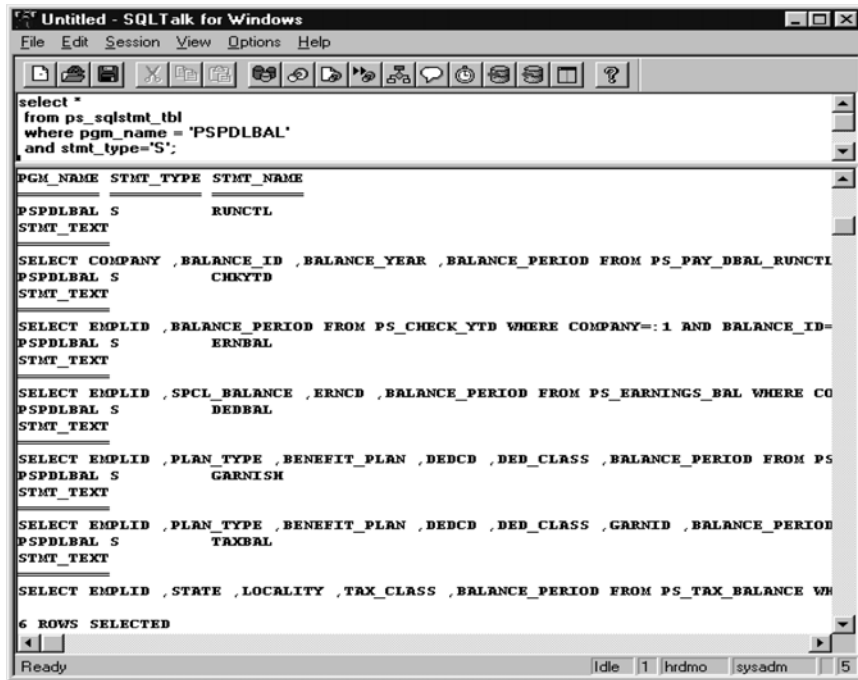


Figure 32.3 Viewing stored SQL statements using SQLTalk

Once the DMS script has executed, the stored SQL statement table can be queried using a simple `Select` statement. Figure 32.3 shows the results of a `Select` when using SQLTalk for Windows. All `Select` statement types for program PSPDLBAL are returned. The COBOL program uses a similar method to retrieve the SQL statement text.

Keep in mind that no need exists to query the stored SQL statements. This is informational only and demonstrates how the SQL statements are stored in the database.

32.3 THE PTPSQLRT MODULE

The module PTPSQLRT performs a variety of functions:

- executes `Select` statements
- fetches rows from the database
- processes SQL updates—`Update`, `Delete`, `Insert` statements
- performs commits and rollbacks
- connects to database
- disconnects from database
- disconnects cursors
- performs error handling

32.3.1 Calling PTPSQLRT

When calling PTPSQLRT from a COBOL program, the following format is used:

```
CALL 'PTPSQLRT' USING action,  
                      sqlrt,  
                      cursor,  
                      statement,  
                      bind-setup,  
                      bind-data,  
                      select-setup,  
                      select-data
```

The preceding parameters are positional and must be passed in the precise order shown. The number of parameters passed to PTPSQLRT can range from two to eight, depending upon the particular Action being executed. For example, an error handling action only requires the first two parameters, while all eight are required when performing an Action that selects data from the database. Once we describe each of the parameters, the requirements for each particular Action will be explained.

32.4 PARAMETER DESCRIPTIONS

We'll now examine each of the parameters that may be passed to the PTPSQLRT module.

32.4.1 Parameter 1—ACTION

A one-character code is used to specify the action to be performed. These codes are already defined in a copybook called PTCSQLRT and should be used when interfacing with PTPSQLRT:

02	ACTION-SELECT	PIC X	VALUE 'S'.
02	ACTION-FETCH	PIC X	VALUE 'F'.
02	ACTION-UPDATE	PIC X	VALUE 'U'.
02	ACTION-COMMIT	PIC X	VALUE 'C'.
02	ACTION-ROLLBACK	PIC X	VALUE 'R'.
02	ACTION-DISCONNECT	PIC X	VALUE 'D'.
02	ACTION-DISCONNECT-ALL	PIC X	VALUE 'A'.
02	ACTION-CONNECT	PIC X	VALUE 'N'.
02	ACTION-ERROR	PIC X	VALUE 'E'.
02	ACTION-CLEAR-STMT	PIC X	VALUE 'L'.
02	ACTION-TRACE	PIC X	VALUE 'T'.
02	ACTION-START-BULK	PIC X	VALUE 'X'.
02	ACTION-STOP-BULK	PIC X	VALUE 'Y'.
02	ACTION-FLUSH-BULK	PIC X	VALUE 'Z'.
02	ACTION-DML-COUNT	PIC X	VALUE 'M'.

Here's how we specify Action as the first parameter in a parameter list:

```
CALL 'PTPSQLRT' USING ACTION-SELECT OF SQLRT  
                      <additional parameters>
```

32.4.2 Parameter 2—SQLRT (Communication Area)

SQLRT is the communication area required by PTPSQLRT. Information about the database and the current run is stored here and passed to PTPSQLRT. The database platform, user ID, password, process instance, and job instance are some examples of data stored here. When control is passed back from PTPSQLRT to the calling module, a return code, which is also found within the PTCSQLRT copybook, is set. This code should be evaluated to determine if the operation were successful:

```
CALL 'PTPSQLRT' USING ACTION-SELECT OF SQLRT
                      SQLRT
                      <additional parameters>

IF RTNCD=ERROR OF SQLRT
  <error handling>
END-IF
```

Here are a few guidelines for using the PTCSQLRT copybook:

SQLRT should be defined as a 01-level item in the working storage section of the main module. A Copy statement should immediately follow, designating PTCSQLRT.

```
/******
*
*          SQL COMMUNICATION
*
*****
01  SQLRT.          COPY PTCSQLRT.
```

The SQLRT communication area must be passed to all called modules. This ensures the same communication area is used:

```
CALL 'PSPDCWS1' USING  SQLRT
                      PSLCT
                      DARRY
```

All called modules must have SQLRT defined as a 01-level item in the programs linkage section. A COPY statement should immediately follow designating PTCSQLRT:

```
LINKAGE SECTION.

/******
*
*          SQL COMMUNICATION
*
*****
01  SQLRT.          COPY PTCSQLRT.
```

The procedure division of the called program must accept the SQLRT parameter:

32.4.3 Parameter 3—CURSOR

Some actions require the use of a database cursor. This is defined as a four-digit computational number. If you don't need to re-use the cursor, the `SQL-CURSOR-COMMON` variable, which is found in the `PTCSQLRT` copybook, may be used. If you need to re-use the cursor, a dedicated variable will need to be defined. A cursor is also referred to as a resource connection unit:

```
CALL 'PTPSQLRT' USING ACTION-SELECT OF SQLRT
                        SQLRT
                        SQL-CURSOR-COMMON OF SQLRT
                        <additional parameters>
```

Here is a brief explanation on what it means to re-use a cursor: If you are selecting from a table only one time in your program, you don't need to dedicate a cursor variable. For example, when you select parameters from a Run Control record, you specify the cursor as `SQL-CURSOR-COMMON` in your `Select` and immediately fetch the row of data for the same cursor. You can then re-use the `SQL-CURSOR-COMMON` variable for other re-useable cursors. Please note that this is simply a convention used by PeopleSoft to make programming easier. It would be unnecessary to define a distinct cursor variable for every SQL function executed only one time.

A dedicated cursor is required when you perform a `Select` and fetch rows multiple times in the COBOL program. Between each fetch from the assigned cursor, you could perform SQL tasks on other open cursors. By dedicating a cursor, you can fetch a row of data at any time in your COBOL program without regard to any other cursors that may currently be open.

32.4.4 Parameter 4—SQL statement name

An SQL statement name is the consolidated name of the stored SQL statement. As we mentioned in our description of the stored SQL statement table, the table keys `PGM_NAME`, `STMT_TYPE`, and `STMT_NAME` must be passed as one string separated by underscores. An example of an SQL statement name is `PSPDLBAL_S_RUNCTL`. PeopleSoft commonly stores this statement name in a variable called `SQL-STMT`. The `SQL-STMT` variable is grouped under a 01-level item with all other components required for the call to `PTPSQLRT`. This may include `Bind Setup/Data`, `Select Setup/Data`, and `SQL Cursor` elements.

The `SQL-STMT` variable in the following example is part of the 01-level item called `S-RUNCTL`:

```
CALL 'PTPSQLRT' USING ACTION-SELECT OF SQLRT
                        SQLRT
```

```
SQL-CURSOR-COMMON OF SQLRT
SQL-STMT OF S-RUNCTL
<additional parameters>
```

32.4.5 Parameter 5—Bind Setup

The fifth and sixth parameters, the Bind Setup and Bind Data parameters, are used together to pass bind variable information. These define the format of the bind variable and the actual bind variable values. The number of bind variables must match that of the stored SQL statement which is designated as :1, :2, etc. The Bind Setup area is a group of picture clauses defined as FILLER, which represents the format of the corresponding bind data. This is known as a setup list. We will discuss this in detail following the parameter descriptions. Let's specify Bind Setup as the fifth parameter in a parameter list.

```
CALL 'PTPSQLRT' USING ACTION-SELECT OF SQLRT
SQLRT
SQL-CURSOR-COMMON OF SQLRT
SQL-STMT OF S-RUNCTL
BIND-SETUP OF S-RUNCTL
<additional parameters>
```

32.4.6 Parameter 6—Bind Data

The sixth parameter is the Bind Data list, which holds the bind variable values used in the stored SQL statement. Each bind data value in the list has a matching bind setup value. Some examples of where bind variables are used are in WHERE clauses or in the values list of an Insert statement:

```
CALL 'PTPSQLRT' USING ACTION-SELECT OF SQLRT
SQLRT
SQL-CURSOR-COMMON OF SQLRT
SQL-STMT OF S-RUNCTL
BIND-SETUP OF S-RUNCTL
BIND-DATA OF S-RUNCTL
<additional parameters>
```

32.4.7 Parameter 7—Select Setup

The seventh and eighth parameters are used together to define the Select area. This Select area is used by the Fetch action to return the selected row of data elements. The Select Setup list is similar to the Bind Setup list. Both use the setup list format, which will be described in detail shortly:

```
CALL 'PTPSQLRT' USING ACTION-SELECT OF SQLRT
SQLRT
SQL-CURSOR-COMMON OF SQLRT
SQL-STMT OF S-RUNCTL
BIND-SETUP OF S-RUNCTL
BIND-DATA OF S-RUNCTL
```

```
SELECT-SETUP OF S-RUNCTL
<additional parameter>
```

32.4.8 Parameter 8—Select Data

The eighth and final parameter is the Select Data list, which holds the returned values from a Fetch action. The number of Select Setup and Select Data entries corresponds to the number of Select columns in the stored SQL statement:

```
CALL 'PTPSQLRT' USING ACTION-SELECT OF SQLRT
                        SQLRT
                        SQL-CURSOR-COMMON OF SQLRT
                        SQL-STMT OF S-RUNCTL
                        BIND-SETUP OF S-RUNCTL
                        BIND-DATA OF S-RUNCTL
                        SELECT-SETUP OF S-RUNCTL
                        SELECT-DATA OF S-RUNCTL
```

Notice the S-RUNCTL designated by the COBOL designator OF for the parameters SQL-STMT, BIND-SETUP, BIND-DATA, SELECT-SETUP, and SELECT-DATA. S-RUNCTL is a 01-level item that contains all of these parameters. Using OF tells the COBOL compiler to qualify the variable names with the 01-level item. This feature allows duplicate variable names to be used. Generally speaking, each 01-level item used to access the database utilizes the same variable names for the desired PTPSQLRT parameters.

32.5 SETUP LISTS

The setup list is used to define the attributes of an accompanying list of data elements. The length of each setup item matches the length of its data item counterpart. The setup item contains a string of characters which designate the data type and, when applicable, the number of decimal places. With the exception of decimal numbers (COMP-3), two codes exist for each data type. If the same data types are defined one after the other, the setup list alternates between the two codes (table 32.2).

Table 32.1 PeopleSoft's Setup List table

Data types	Codes	Length	Data list picture
Character	C, H	1 to 255	X(1) through X(255)
Date	D, A	10	X(10)
Time	T, E	26	X(26)
Small integer	S, M	2	[S]999 or [S]9999 COMP
Large integer	I, N	4	[S]9(8) or [S]9(9) COMP

Table 32.1 PeopleSoft's Setup List table (continued)

Data types	Codes	Length	Data list picture
Decimal number	d[P...]	1 to 8	[S]9(w)[V9(d)] COMP-3 Example 1: S9(5)V9(2) COMP-3 => 2PPP Example 2: 999V99 COMP-3 => 2PP Example 3: S9(11)V999 COMP-3 => 3PPPPPPP
END OF LIST	Z	1	All setup and data areas must be terminated with the character 'Z'.

Note that all Bind-Setup, Bind-Data, Select-Setup, and Select-Data areas must end with the termination character of 'Z.' If there are no bind data values, as in the case of an unconditional Select or Update, a single 'Z' termination character must exist in the Bind-Setup/Bind-Data areas.

Let's elaborate briefly on the decimal number setup (COMP-3). Look at the three examples used in the Decimal Number section of the Setup List table (table 32.1). The first example requires a string defined as PIC X(4) with a value of 2PPP. This represents the number of decimal places along with the total length of the data field. A field defined as S9(5)V99 COMP-3 takes up four bytes. The trailing 'P' character simply fills the remainder of the field after the number of decimal places. The second example would use a string defined as PIC X(3) with a value of 2PP, which represents two decimals and a total length of three bytes. The data field in example 2 is defined as 999V99 which takes up three bytes. The last example uses a PIC X(8) string with a value of 3PPPPPPP. This designates three decimal places and a total length of eight bytes. The data field defined as S9(11)V999 COMP-3 takes up eight bytes in storage.

Let's take a look at an actual working storage area that includes a setup list. This section was taken from the PSPDLBAL module. Notice the stored SQL statement definition, Bind Setup/Data, and Select Setup/Data areas:

```

01  S-RUNCTL.
    02  SQL-STMT                                PIC X(18)  VALUE
                                                'PSPDLBAL_S_RUNCTL'.

    02  BIND-SETUP.
        03  FILLER                                PIC X(8)   VALUE ALL 'C'.
        03  FILLER                                PIC X(30)  VALUE ALL 'H'.
        03  FILLER                                PIC X    VALUE 'Z'.

    02  BIND-DATA.
        03  OPRID                                PIC X(8) .
        03  BATCH-RUN-ID                        PIC X(30) .
        03  FILLER                                PIC X    VALUE 'Z'.

```

```

02  SELECT-SETUP.
    03  FILLER                PIC X(10)  VALUE ALL 'C'.
    03  FILLER                PIC XX     VALUE ALL 'H'.
    03  FILLER                PIC XX     VALUE ALL 'S'.
    03  FILLER                PIC XX     VALUE ALL 'M'.
    03  FILLER                PIC X      VALUE 'Z'.

02  SELECT-DATA.
    03  COMPANY               PIC X(10).
    03  BALANCE-ID            PIC XX.
    03  BALANCE-YEAR          PIC 9999          COMP.
    03  BALANCE-PERIOD        PIC 999          COMP.
    03  FILLER                PIC X      VALUE 'Z'.

```

The preceding example shows a good sample of contiguous setup list strings. Let's examine the Bind-Setup and Bind-Data areas. The Bind-Data area is for the OPRID and BATCH-RUN-ID, which are both character fields. The Bind-Setup uses the character 'C' for the OPRID setup list string while the character 'H' is used for the BATCH-RUN-ID setup list string. You can also see that the character 'Z' terminates each setup and data area. This produces an image that will be recognized by the PTPSQLRT module to determine the position of the two fields.

If both fields use 'C' in the setup list string, the PTPSQLRT module interprets this as one thirty-eight character field with the second field missing. This produces an error in the COBOL program.

Let's also look at the stored SQL statement that will be executed. The following displays the portion of the DMS script containing the statement text:

```

STORE PSPDLBAL_S_RUNCTL
SELECT COMPANY,
        BALANCE_ID,
        BALANCE_YEAR,
        BALANCE_PERIOD
FROM PS_PAY_DBAL_RUNCTL
WHERE OPRID      = :1
      AND RUN_CNTL_ID = :2
;

```

The Bind Setup/Data area in working storage contains the OPRID and BATCH-RUN-ID and will be used as the criteria for the Select. PTPSQLRT substitutes these for bind variables :1 and :2. The Select Setup/Data area will be used to store the results of the Select statement. COMPANY, BALANCE_ID, BALANCE_YEAR, and BALANCE_PERIOD will be stored in the format specified in the Select Setup. It is the developers responsibility to ensure that the datatypes are compatible. The Select Data elements are now populated and can be used by the COBOL program.

32.6 ACTION REQUIREMENTS

Let's review some of the basic actions found in PeopleSoft COBOL and the required parameters for each.

CONNECT— connects to the database.

```
CALL 'PTPSQLRT' USING ACTION-CONNECT OF SQLRT
                        SQLRT
                        SQL-CURSOR-COMMON OF SQLRT

IF RTNCD-ERROR OF SQLRT
    <Error Handling>
END-IF
```

CONNECT uses three parameters: The Action, SQL communication area, which is the minimum requirement for all actions, and a reusable cursor which is required to connect. The return code is checked to determine whether or not the connect action was successful.

DISCONNECT— disconnects a cursor from database.

DISCONNECT uses three parameters. Notice the following example specifies a dedicated cursor, defined within the S-PYGRP 01-level of working storage.

```
CALL 'PTPSQLRT' USING ACTION-DISCONNECT OF SQLRT
                        SQLRT
                        SQL-CURSOR OF S-PYGRP

IF RTNCD-ERROR OF SQLRT
    <Error Handling>
END-IF
```

DISCONNECT ALL— disconnects all cursors from database.

```
CALL 'PTPSQLRT' USING ACTION-DISCONNECT-ALL OF SQLRT
                        SQLRT

IF RTNCD-ERROR OF SQLRT
    <Error Handling>
END-IF
```

DISCONNECT ALL uses two parameters. All cursors are disconnected. You will find this immediately before the end of the program.

ERROR—is the error handling routine.

```
CALL 'PTPSQLRT' USING ACTION-CONNECT OF SQLRT
                        SQLRT
                        SQL-CURSOR-COMMON OF SQLRT
```

```

IF RTNCD=ERROR OF SQLRT

    MOVE 'SELECT-RUNCTL(CONNECT)' TO ERR-SECTION OF SQLRT
    PERFORM ZZ000-SQL-ERROR
END-IF

...

ZZ000-SQL-ERROR SECTION.
ZZ000.

CALL 'PTPSQLRT' USING ACTION-ERROR OF SQLRT
                     SQLRT

```

ACTION-ERROR—uses two parameters: the Action and SQL communication area. This Action provides a consistent means of error handling. The PTCSQLRT copybook contains a field called ERR-SECTION. The section which caused the error should be placed in the ERR-SECTION field. The previous example shows both the controlling section, which may cause an error, and the ZZ000-SQL-ERROR section, which executes the Action-Error process. The error handling procedure displays the section which caused the error and also halts further processing. The user can see that the error occurred in the SELECT-RUNCTL section while trying to connect to the database.

COMMIT—performs a commit.

```

CALL 'PTPSQLRT' USING ACTION-COMMIT OF SQLRT
                     SQLRT
                     SQL-CURSOR-COMMON OF SQLRT
IF RTNCD=ERROR OF SQLRT

    MOVE 'COMMIT' TO ERR-SECTION OF SQLRT
    PERFORM ZZ000-SQL-ERROR
END-IF

```

ACTION-COMMIT—uses three parameters: The Action, SQL Communication Area, and a database cursor. The SQL-CURSOR-COMMON variable (found in PTCSQLRT) may be used for reusable cursors. When executed, all work will be committed since the latest commit (or rollback).

ROLLBACK—performs a Rollback.

```

CALL 'PTPSQLRT' USING ACTION-ROLLBACK OF SQLRT
                     SQLRT
                     SQL-CURSOR-COMMON OF SQLRT
IF RTNCD=ERROR OF SQLRT

    MOVE 'ROLLBACK' TO ERR-SECTION OF SQLRT
    PERFORM ZZ000-SQL-ERROR
END-IF

```

ACTION-ROLLBACK—uses three parameters: The Action, SQL Communication Area, and a database cursor. When executed, all work completed since the last commit will be rolled back.

SELECT—selects and formats data from the database.

```
MOVE OPRID          OF SQLRT  TO  OPRID          OF S-RUNCTL
MOVE BATCH-RUN-ID  OF SQLRT  TO  BATCH-RUN-ID  OF S-RUNCTL

CALL 'PTPSQLRT' USING  ACTION-SELECT OF SQLRT
                        SQLRT
                        SQL-CURSOR-COMMON OF SQLRT
                        SQL-STMT OF S-RUNCTL
                        BIND-SETUP OF S-RUNCTL
                        BIND-DATA OF S-RUNCTL
                        SELECT-SETUP OF S-RUNCTL
                        SELECT-DATA OF S-RUNCTL

IF RTNCD-ERROR OF SQLRT

    MOVE 'SELECT-RUNCTL(SELECT)' TO ERR-SECTION OF SQLRT
    PERFORM ZZ000-SQL-ERROR
END-IF
```

ACTION-SELECT—uses all eight available parameters. The primary function of ACTION-SELECT is to create a result set of data from the database. Once created, the rows may be retrieved one-at-a-time using a Fetch action which we'll explain next. The example above is used to select Run Control information and is using a reuseable cursor. Before the call to PTPSQLRT, the OPRID and BATCH-RUN-ID are moved to the Bind-Data area within the S-RUNCTL 01-level area. The Bind Data is used as the Where clause criteria in the SQL statement. Let's look at some of the parameter definitions in working storage:

```
01  S-RUNCTL.
    02  SQL-STMT                      PIC X(18)    VALUE
                                           'PSPDLBAL_S_RUNCTL'.

    02  BIND-SETUP.
        03  FILLER                    PIC X(8)     VALUE ALL 'C'.
        03  FILLER                    PIC X(30)    VALUE ALL 'H'.
        03  FILLER                    PIC X        VALUE 'Z'.

    02  BIND-DATA.
        03  OPRID                    PIC X(8) .
        03  BATCH-RUN-ID             PIC X(30) .
        03  FILLER                    PIC X        VALUE 'Z'.

    02  SELECT-SETUP.
        03  FILLER                    PIC X(10)    VALUE ALL 'C'.
```



```

03 FILLER PIC XX VALUE ALL 'H'.
03 FILLER PIC XX VALUE ALL 'S'.
03 FILLER PIC XX VALUE ALL 'M'.
03 FILLER PIC X VALUE 'Z'.

02 SELECT-DATA.
03 COMPANY PIC X(10).
03 BALANCE-ID PIC XX.
03 BALANCE-YEAR PIC 9999 COMP.
03 BALANCE-PERIOD PIC 999 COMP.
03 FILLER PIC X VALUE 'Z'.

```

We can see the working storage definitions used in the ACTION-SELECT example. Notice the last five parameters in the call are grouped together under the same 01-level item called S-RUNCTL. If a dedicated cursor were required, it would also be defined in the S-RUNCTL area. This is a very structured and consistent approach used throughout PeopleSoft COBOL. Any analysis or modifications may be carried out with relative ease due to this structure.

Let's take a closer look at the SQL-STMT parameter. This contains the consolidated key of the SQL statement stored in the SQL statement table. Let's look at the SQL statement text retrieved by PTPSQLRT when the ACTION-SELECT is performed:

```

SELECT COMPANY,
       BALANCE_ID,
       BALANCE_YEAR,
       BALANCE_PERIOD
FROM PS_PAY_DBAL_RUNCTL
WHERE OPRID      =:1
      AND RUN_CNTL_ID =:2

```

The bind data OPRID and BATCH-RUN-ID are substituted for the bind variables :1 and :2. The statement is executed, and Select Data will be used to accept the data. An ACTION-FETCH needs to be performed to physically retrieve each row in the result set created by ACTION-SELECT.

FETCH—fetches a single row from result set created by Select:

```

MOVE OPRID      OF SQLRT TO OPRID      OF S-RUNCTL
MOVE BATCH-RUN-ID OF SQLRT TO BATCH-RUN-ID OF S-RUNCTL

CALL 'PTPSQLRT' USING ACTION-SELECT OF SQLRT
                      SQLRT
                      SQL-CURSOR-COMMON OF SQLRT
                      SQL-STMT OF S-RUNCTL
                      BIND-SETUP OF S-RUNCTL
                      BIND-DATA OF S-RUNCTL
                      SELECT-SETUP OF S-RUNCTL
                      SELECT-DATA OF S-RUNCTL

```

```

IF RTNCD=ERROR OF SQLRT

    MOVE 'SELECT-RUNCTL(SELECT)' TO ERR-SECTION OF SQLRT
    PERFORM ZZ000-SQL-ERROR
END-IF

INITIALIZE SELECT-DATA OF S-RUNCTL

CALL 'PTPSQLRT' USING ACTION-FETCH OF SQLRT
                     SQLRT
                     SQL-CURSOR-COMMON OF SQLRT

IF RTNCD=ERROR OF SQLRT

    IF RTNCD-END OF SQLRT

        DISPLAY 'Delete Balances Run Control Missing.'
        DISPLAY ' for Operator ID ' OPRID OF S-RUNCTL
        DISPLAY ' and Batch Run ID ' BATCH-RUN-ID OF S-RUNCTL
        SET RTNCD=USER-ERROR OF SQLRT TO TRUE
        PERFORM ZZ000-SQL-ERROR
    ELSE
        MOVE 'SELECT-RUNCTL(FETCH)' TO ERR-SECTION OF SQLRT
        PERFORM ZZ000-SQL-ERROR
    END-IF
ELSE
    PERFORM DD000-RUNCTL-ACCEPTED
END-IF

```

ACTION-FETCH—uses three parameters. When the Fetch is performed, the data are placed in the Setup Data area defined in the ACTION-SELECT for the designated cursor. Notice that the ACTION-SELECT above utilizes the SQL-CURSOR-COMMON reusable cursor. The ACTION-FETCH uses the same cursor and all associated characteristics including the Select Data area. Upon returning from the Fetch, the return code is tested. If there is no error, a row has been successfully returned. If there is an error, it could be due to an end-of-data condition. An error message is displayed if there is no data (Missing Run Control). Any other database errors are handled as well with a simple 'SELECT-RUNCTL(FETCH)' message.

Our example was very straightforward. Since we are selecting a Run Control record, we are expecting one row to be returned. If multiple rows were processed, a loop would be required. The rows would be fetched one at a time with an end-of-data test used to break out of the loop. The Select Data area would be updated with each fetched row and utilized accordingly by the program.

UPDATE—performs an Insert, Update, or Delete.

```

MOVE OPRID          OF SQLRT TO OPRID          OF D-RUNCTL
MOVE BATCH-RUN-ID OF SQLRT TO BATCH-RUN-ID OF D-RUNCTL

```

```

CALL 'PTPSQLRT' USING ACTION-UPDATE OF SQLRT
                     SQLRT
                     SQL-CURSOR-COMMON OF SQLRT
                     SQL-STMT OF D-RUNCTL
                     BIND-SETUP OF D-RUNCTL
                     BIND-DATA OF D-RUNCTL

IF RTNCD-ERROR OF SQLRT

    MOVE 'RUNCTL-ACCEPTED(DELETE)' TO ERR-SECTION OF SQLRT
    PERFORM ZZ000-SQL-ERROR
END-IF

PERFORM ZA000-COMMIT

```

ACTION-UPDATE—uses six parameters. This function is used to execute Inserts, Updates, and Deletes. The Select Setup and Data areas are omitted. The Bind Setup and Data areas are used to pass WHERE clause criteria, Update values and Insert values. The order of the Bind Setup/Data lists must match the order of the bind variables (:1, :2, :3, etc.) in the stored SQL statement. Now, let's have a look at the working storage section:

```

01 D-RUNCTL.
    02 SQL-STMT                                PIC X(18)    VALUE
                                                'PSPDLBAL_D_RUNCTL'.

    02 BIND-SETUP.
        03 FILLER                                PIC X(8)    VALUE ALL 'C'.
        03 FILLER                                PIC X(30)   VALUE ALL 'H'.
        03 FILLER                                PIC X      VALUE 'Z'.

    02 BIND-DATA.
        03 OPRID                                PIC X(8).
        03 BATCH-RUN-ID                        PIC X(30).
        03 FILLER                                PIC X      VALUE 'Z'.

```

We see the 01-level item D-RUNCTL, which is the area used in our ACTION-UPDATE example. Notice the name of the SQL-STMT, 'PSPDLBAL_D_RUNCTL'. The middle character indicates that this is a Delete statement. The only bind variables utilized by a Delete are in the WHERE clause. We now know that the criteria for the Delete is OPRID and BATCH-RUN-ID. This statement will delete the Run Control record for the process we're running.

```

DELETE
FROM PS_PAY_DBAL_RUNCTL
WHERE OPRID      =:1
AND RUN_CNTL_ID =:2

```

Bind variables :1 and :2 serve
 as the WHERE clause criteria in
 a DELETE

If you look in the DMS script (PSPDLBAL.DMS) or query the SQL statement table itself, you find the SQL statement displayed previously. The OPRID and BATCH-RUN-ID values in the Bind Data area are substituted for the :1 and :2 bind variables. The Delete SQL statement is then compiled and executed, and the Run Control record is deleted (if all goes well). Notice the error handling in our example as well as the Commit routine, which is performed if the Action were successful.

Let's look at a sample of an Update and Insert statement. We'll examine the working storage area and the portion of the DMS script which contains the SQL statement text. These examples can be found in the program PAPPPYMT.CBL and the DMS script PAPPPYMT.DMS:

```

01  U-RUNCNTL.
    05  SQL-CURSOR                PIC S9(4)    VALUE 0      COMP.
    05  SQL-STMT                  PIC X(18)    VALUE
                                     'PAPPPYMT_U-RUNCNTL'.

    05  BIND-SETUP.
        10  FILLER                PIC X(11)    VALUE ALL 'C'.
        10  FILLER                PIC X(8)     VALUE ALL 'H'.
        10  FILLER                PIC X(30)    VALUE ALL 'C'.
        10  FILLER                PIC X(01)    VALUE 'Z'.

    05  BIND-DATA.
        10  EMPLID                PIC X(11)    VALUE SPACE.
        10  OPRID                 PIC X(8)     .
        10  RUN-CNTL-ID           PIC X(30)    .
        10  FILLER                PIC X(01)    VALUE 'Z'.

```

The preceding example displays a typical working storage area used in an Update action. Depending on the SQL statement, the bind data can be any combination of Update values or WHERE clause criteria. The bind data may be all Update values with criteria hard-coded in the SQL statement itself (or no criteria at all for a mass update). The bind data may be made up entirely of WHERE clause criteria with the Update value hard-coded in the SQL statement. There may not be any bind data values at all! Consider the statement, 'UPDATE PS_INSTALLATION SET EMPLID_LAST_EMPL = 0'. There are no bind data values and no WHERE clause values. The SQL statement requires no bind data at all. The bind data and bind setup lists would still be required. A single termination character of 'Z' would reside in both lists, and the statement would be executed without bind values.

```

STORE PAPPPYMT_U-RUNCNTL
UPDATE PS_PA_RUN_CNTL
    SET EMPLID      = :1
    WHERE OPRID     = :2
    AND RUN_CNTL_ID = :3

```

Bind variable :1 is used as an UPDATE value while :2 and :3 are used as WHERE clause criteria.

The bind data values in our (Update) working storage example correspond to the bind variables :1, :2, and :3 in the SQL statement text depicted above. Once again,

the order and datatype of the bind data should match that of the bind variables in the SQL statement text.

Now let's look at a typical working storage area used in an Insert action. The bind values EMPLID, BENEFIT_PLAN, EFFDT, and PENSION-STATUS will be inserted into the table contained in the SQL statement:

```

01  I-PENSTAT.
    05  SQL-CURSOR          PIC S9(4)  VALUE 0      COMP.
    05  SQL-STMT            PIC X(18)   VALUE
                                'PAPPPYMT_I_PENSTAT' .

    05  BIND-SETUP.
        10  FILLER          PIC X(11)   VALUE ALL 'C' .
        10  FILLER          PIC X(6)    VALUE ALL 'H' .
        10  FILLER          PIC X(10)   VALUE ALL 'D' .
        10  FILLER          PIC X(3)    VALUE ALL 'C' .
        10  FILLER          PIC X(01)   VALUE 'Z' .

    05  BIND-DATA.
        10  EMPLID          PIC X(11) .
        10  BENEFIT-PLAN    PIC X(6) .
        10  EFFDT          PIC X(10) .
        10  PENSION-STATUS  PIC X(3) .
        10  FILLER          PIC X(01)   VALUE 'Z' .

```

The bind variables :1 thru :4 will be replaced by the bind data values in the previous working storage definition. As you know, the order and datatype of the bind variables and bind data must match exactly to be executed successfully:

```

STORE PAPPPYMT_I_PENSTAT
INSERT INTO PS_PA_EMP_PEN_STAT
    (EMPLID
    , BENEFIT_PLAN
    , EFFDT
    , PENSION_STATUS)
VALUES (:1, :2, :3, :4)

```

Bind variables :1 thru :4 must be accounted for properly in the Bind Setup and Bind Data areas of the COBOL program

KEY POINTS

- 1 It is always wise to avoid COBOL customizations whenever possible. Make sure all possible solutions are investigated before deciding to modify delivered COBOL processes.
- 2 Even if you don't plan on modifying COBOL programs, become familiar with PeopleSoft COBOL techniques. There will surely be times when you need to browse the COBOL source code when troubleshooting.
- 3 The PTPSQLRT module regulates all database activity required by the COBOL process. Since it is one encapsulated routine, it remains consistent across all database platforms.
- 4 All SQL statements are stored in a database table and retrieved by the COBOL process where they are compiled and executed.
- 5 The SQL statements are loaded into the stored SQL statement table using Data Mover scripts. Any required modifications should be made to the scripts so they may be reloaded.
- 6 PTPSQLRT performs database functions as well as error handling. These functions are referred to as actions. Up to eight positional parameters may be passed, depending on the action requested.
- 7 The eight parameters are the Action, the SQL communication area, an SQL cursor, the SQL statement name reference, the Bind Setup/Data areas and the Select Setup/Data areas.
- 8 The copybook PTCSQLRT contains the SQL communication area used by the PTPSQLRT module and must be included in all COBOL modules. The Main module must define the SQLRT area as an 01-level item in working storage and must pass this area as a parameter to other called modules. All called or subordinate modules must define the SQLRT area as an 01-level item in the linkage section and the procedure division must accept the SQLRT parameter. This insures the same SQLRT area is used by all modules.

KEY POINTS (CONTINUED)

- 9 Setup lists are used to define the Bind and Setup areas in working storage. Each datatype has a pair of corresponding setup codes. The codes are alternated when two consecutive fields with the same datatype are used. This allows the PTPSQLRT module to parse the incoming bind data and outgoing select data properly.



C H A P T E R 3 3

Modifying PeopleSoft COBOL

- 33.1 Defining a modification 738
- 33.2 Making our modifications 739

33.1 *DEFINING A MODIFICATION*

To demonstrate how to modify a PeopleSoft COBOL program, we need to define a task. We'll continue to use the program PSPDLBAL as a model. As we mentioned earlier, this process is used in PeopleSoft Payroll to delete obsolete balances from the system. Let's take a closer look at this process before we decide on a sample customization.

33.1.1 *Delivered functionality*

In short, the PSPDLBAL process deletes obsolete balances from the following tables:

- YTD Check Balances (CHECK_YTD)
- YTD Earnings Balances (EARNINGS_BAL)

- YTD Deduction Balances (DEDUCTION_BAL)
- YTD Garnishment Balances (GARN_BALANCE)
- YTD Tax Balances (TAX_BALANCE)

The Run Control record contains the parameters:

- Company
- Balance Year
- Balance ID
- Balance Period

Any obsolete balances in the tables for the Company and Balance ID that were before (or equal to) the period defined by the Balance Year and Balance Period will be removed. All five tables will be checked and updated where necessary.

33.1.2 A simple modification

Let's say that I would like to have the ability to delete balances from all of the tables (as delivered) OR only one of the tables if I so choose. To accomplish this, I'm going to add the field RECNAME to the Run Control record. If no recname is specified on the Run Control panel, then all five tables will be processed. If the RECNAME is not blank, then I'll only process the record specified. The valid values are CHECK_YTD, EARNINGS_BAL, DEDUCTION_BAL, GARN_BALANCE, and TAX_BALANCE.

If a value is entered that is not blank or not one of these five tables, an error message should be produced and the process halted.

Please note that we'll skip the steps to add the RECNAME field to the Run Control record and panel. A full description of adding fields to records and panels using Application Designer has been given earlier in this book. We'll assume these changes have been implemented during our modification example.

33.2 MAKING OUR MODIFICATIONS

Let's take a look at the AA000-MAIN section following the PROCEDURE DIVISION. This is where the routine for each individual table is unconditionally called and will be one of the sections we'll modify. Before each of these routines is performed, we're going to test the RECNAME parameter passed on the Run Control record. If it is blank or matches the tablename, then the routine will be performed:

```

/*****
*
*   PROCEDURE DIVISION.
*
*****
*
*   AA000-MAIN SECTION.
*   AA000.
*

```

COPY PTCLIBFX.
COPY PSCVERSN.

SET PAYROLL-STEP-DLTBALNC OF PSLCT TO TRUE
PERFORM DA000-SELECT-RUNCTL

ACCEPT TIME-OUT OF W-WK FROM TIME
INSPECT TIME-OUT OF W-WK CONVERTING SPACE TO ':'
INSPECT TIME-OUT OF W-WK CONVERTING '/' TO '.'
DISPLAY 'Delete Balances started for Company: '
COMPANY OF S-RUNCTL
DISPLAY ' Calendar Year: '
BALANCE-YEAR OF S-RUNCTL
DISPLAY ' Month: '
BALANCE-PERIOD OF S-RUNCTL
DISPLAY ' at ' TIME-OUT OF W-WK '.'

PERFORM GA000-PURGE-CHECK-YTD
PERFORM IA000-PURGE-EARNINGS-BAL
PERFORM KA000-PURGE-DEDUCTION-BAL
PERFORM MA000-PURGE-GARN-BALANCE
PERFORM OA000-PURGE-TAX-BALANCE
PERFORM SA000-TERM

**Each PURGE routine is
called unconditionally**

COPY PSCRTNCD.

.
MAIN-EXIT.
STOP RUN.

Let's also take a look at the working storage area used when selecting data from the Run Control record:

```

/*****
*          PAY_DBAL_RUNCTL BUFFER AND STMT          *
*****/
01  S-RUNCTL.
    02  SQL-STMT                      PIC X(18)    VALUE
                                           'PSPDLBAL_S_RUNCTL' .

    02  BIND-SETUP.
        03  FILLER                    PIC X(8)     VALUE ALL 'C' .
        03  FILLER                    PIC X(30)    VALUE ALL 'H' .
        03  FILLER                    PIC X        VALUE 'Z' .

    02  BIND-DATA.
        03  OPRID                     PIC X(8) .
        03  BATCH-RUN-ID              PIC X(30) .

```

```

03 FILLER PIC X VALUE 'Z'.

02 SELECT-SETUP.
03 FILLER PIC X(10) VALUE ALL 'C'.
03 FILLER PIC XX VALUE ALL 'H'.
03 FILLER PIC XX VALUE ALL 'S'.
03 FILLER PIC XX VALUE ALL 'M'.
03 FILLER PIC X VALUE 'Z'.

02 SELECT-DATA.
03 COMPANY PIC X(10).
03 BALANCE-ID PIC XX.
03 BALANCE-YEAR PIC 9999 COMP.
03 BALANCE-PERIOD PIC 999 COMP.
03 FILLER PIC X VALUE 'Z'.

```

Take a close look at the Select Setup and Data areas as presented. We need to modify both of these areas to accept our new field RECNAME. Let's do it now:

```

/*****
*          PAY_DBAL_RUNCTL BUFFER AND STMT          *
*****/
01 S-RUNCTL.
02 SQL-STMT PIC X(18) VALUE
    'PSPDLBAL_S_RUNCTL'.

02 BIND-SETUP.
03 FILLER PIC X(8) VALUE ALL 'C'.
03 FILLER PIC X(30) VALUE ALL 'H'.
03 FILLER PIC X VALUE 'Z'.

02 BIND-DATA.
03 OPRID PIC X(8).
03 BATCH-RUN-ID PIC X(30).
03 FILLER PIC X VALUE 'Z'.

02 SELECT-SETUP.
03 FILLER PIC X(10) VALUE ALL 'C'.
03 FILLER PIC XX VALUE ALL 'H'.
03 FILLER PIC XX VALUE ALL 'S'.
03 FILLER PIC XX VALUE ALL 'M'.
03 FILLER PIC X(15) VALUE ALL 'C'.
03 FILLER PIC X VALUE 'Z'.

02 SELECT-DATA.
03 COMPANY PIC X(10).
03 BALANCE-ID PIC XX.
03 BALANCE-YEAR PIC 9999 COMP.
03 BALANCE-PERIOD PIC 999 COMP.
03 RECNAME PIC X(15).
03 FILLER PIC X VALUE 'Z'.

```

RECNAME
picture string
added to
Select
Setup list

RECNAME
added to Select
Data list

The preceding example shows the changes we've made to the Select Setup and Data area. The RECNAME field length is fifteen characters. We've added the string 'CCCCCCCCCCCCCCC' as filler to the setup list. We've also added the field RECNAME to the data list. We're now ready to accept the additional parameter RECNAME on the Run Control record. We still have one important step to complete if we're going to pass an additional Run Control parameter. We need to update the DMS script to select the new field:

```
STORE PSPDLBAL_S_RUNCTL
SELECT
  COMPANY
  ,BALANCE_ID
  ,BALANCE_YEAR
  ,BALANCE_PERIOD
FROM PS_PAY_DBAL_RUNCTL
WHERE OPRID=:1
      AND RUN_CNTL_ID=:2
```

Now we can see the portion of the DMS script in its delivered form. We still need to add the new RECNAME column to the select list. It needs to be the last field contained in the SELECT list so it matches the order used in the Select Setup and Data areas in working storage. Let's change it now:

```
STORE PSPDLBAL_S_RUNCTL
SELECT
  COMPANY
  ,BALANCE_ID
  ,BALANCE_YEAR
  ,BALANCE_PERIOD
  ,RECNAME
FROM PS_PAY_DBAL_RUNCTL
WHERE OPRID=:1
      AND RUN_CNTL_ID=:2
```

RECNAME column added
to SELECT list

We can see that the new column RECNAME has been added to the end of the Select list in the DMS script. Data Mover should be used to execute the DMS script so the stored SQL statement table will have the modified version of the SQL statement. Let's now make some programming changes to the delivered process:

Listing 33.1

Main section of PSPDLBAL.cbl after modifications

```
/* *****
*
*   PROCEDURE DIVISION.
*
* *****
```

```

*
AA000-MAIN SECTION.
AA000.
*
*****

COPY PTCLIBFX.
COPY PSCVERSN.

SET PAYROLL-STEP-DLTBALNC OF PSLCT TO TRUE
PERFORM DA000-SELECT-RUNCTL

ACCEPT TIME-OUT OF W-WK FROM TIME
INSPECT TIME-OUT OF W-WK CONVERTING SPACE TO ':'
INSPECT TIME-OUT OF W-WK CONVERTING '/' TO '.'
DISPLAY 'Delete Balances started for Company: '
      COMPANY OF S-RUNCTL
DISPLAY '          Calendar Year: '
      BALANCE-YEAR OF S-RUNCTL
DISPLAY '          Month: '
      BALANCE-PERIOD OF S-RUNCTL
DISPLAY ' at ' TIME-OUT OF W-WK '.'
```

* Modification - Validate RECNAME Parameter

```

IF RECNAME OF S-RUNCTL NOT EQUAL SPACE
AND RECNAME OF S-RUNCTL NOT EQUAL 'CHECK_YTD'
AND RECNAME OF S-RUNCTL NOT EQUAL 'EARNINGS_BAL'
AND RECNAME OF S-RUNCTL NOT EQUAL 'DEDUCTION_BAL'
AND RECNAME OF S-RUNCTL NOT EQUAL 'GARN_BALANCE'
AND RECNAME OF S-RUNCTL NOT EQUAL 'TAX_BALANCE'

      DISPLAY 'Invalid RECNAME: '
      RECNAME OF S-RUNCTL
      MOVE 'MAIN(RECNAME)' TO ERR-SECTION OF SQLRT
      PERFORM ZZ000-SQL-ERROR

END-IF
```

* Modification - Conditionally perform each routine

```

IF RECNAME OF S-RUNCTL = SPACE
OR RECNAME OF S-RUNCTL = 'CHECK_YTD'
      PERFORM GA000-PURGE-CHECK-YTD
END-IF

IF RECNAME OF S-RUNCTL = SPACE
OR RECNAME OF S-RUNCTL = 'EARNINGS_BAL'
      PERFORM IA000-PURGE-EARNINGS-BAL
END-IF

IF RECNAME OF S-RUNCTL = SPACE
```

Simple validation routine added. If RECNAME is not one of six valid values, an error message is produced, and the process is halted.

```

OR RECNAME OF S-RUNCTL = 'DEDUCTION_BAL'
  PERFORM KA000-PURGE-DEDUCTION-BAL
END-IF

IF RECNAME OF S-RUNCTL = SPACE
OR RECNAME OF S-RUNCTL = 'GARN_BALANCE'
  PERFORM MA000-PURGE-GARN-BALANCE
END-IF

IF RECNAME OF S-RUNCTL = SPACE
OR RECNAME OF S-RUNCTL = 'TAX_BALANCE'
  PERFORM OA000-PURGE-TAX-BALANCE
END-IF

PERFORM SA000-TERM

COPY PSCRTNCD.

.
MAIN-EXIT.
STOP RUN.

```

Conditional logic has been added to control the execution of each of the five purge routines. They will be performed if the **RECNAME** is not entered (**SPACE**) or matches the table updated by each particular routine.

Consider the programming modifications we've implemented. The changes, although not elegant by any means, effectively produce the results we want. The validation of the **RECNAME** parameter could have been performed within the **DA000-Select-Runctl** section. We placed this validation within the **AA000-Main** section to keep things simple. After all, the focus of this exercise is on accessing an additional field in the database, not on creating the ideal placement of standard COBOL code.

A series of simple **IF** statements controls the execution of each of the Purge routines. As you can see, the changes required to access the new field **RECNAME** have been minimal. Let's review the basic steps required to implement our change:

- 1 Add **RECNAME** field to the Run Control record and panel using Application Designer.
- 2 Add **RECNAME** column to the **Select** list in the DMS script. Run Data Mover to update the stored SQL statement table with the new version of the SQL statement.
- 3 Update the **Select Setup** and **Data** areas in the working storage section to accept the new **RECNAME** parameter.
- 4 Add any required programming logic that utilizes the new **RECNAME** field. In our example, this includes validating parameters along with controlling the balance updates using the new parameter.

33.2.1 One important note

Because of PeopleSoft's structured technique, it isn't necessary to modify the routine that actually selects the Run Control information. In our example, this routine is called **DA000-Select-Runctl**. The only modification required within the program to accept an

additional field is to the Select Setup and Select Data areas. Because the call to PTPSQLRT uses both Select Setup and Select Data as input parameters, the new field automatically is included. Let's look at the Run Control access routine:

Listing 33.2

DA000-SELECT-RUNCTL section (no modifications required)

```

/*****
*
DA000-SELECT-RUNCTL SECTION.
DA000.
*
*****/

CALL 'PTPSQLRT' USING ACTION-CONNECT OF SQLRT
                     SQLRT
                     SQL-CURSOR-COMMON OF SQLRT
IF RTNCD-ERROR OF SQLRT

    MOVE 'SELECT-RUNCTL(CONNECT)' TO ERR-SECTION OF SQLRT
    PERFORM ZZ000-SQL-ERROR
END-IF

IF PROCESS-INSTANCE OF SQLRT NOT = ZERO

    PERFORM DB000-SET-RUN-STAT-PROCESSING
    MOVE PROCESS-INSTANCE OF SQLRT
      TO PROCESS-INSTANCE-ERRMSG OF PSLCT
ELSE

    CALL 'PTPRUNID' USING SQLRT
                        PROCESS-INSTANCE-ERRMSG
                        OF W-PRC-INSTANCE
    MOVE PROCESS-INSTANCE-ERRMSG OF W-PRC-INSTANCE
      TO PROCESS-INSTANCE-ERRMSG OF PSLCT
END-IF

MOVE OPRID OF SQLRT TO OPRID OF S-RUNCTL
MOVE BATCH-RUN-ID OF SQLRT TO BATCH-RUN-ID OF S-RUNCTL

CALL 'PTPSQLRT' USING ACTION-SELECT OF SQLRT
                     SQLRT
                     SQL-CURSOR-COMMON OF SQLRT
                     SQL-STMT OF S-RUNCTL
                     BIND-SETUP OF S-RUNCTL
                     BIND-DATA OF S-RUNCTL
                     SELECT-SETUP OF S-RUNCTL
                     SELECT-DATA OF S-RUNCTL

IF RTNCD-ERROR OF SQLRT

```

RECNAME is added within Select-Setup and Select-Data areas in working storage. RECNAME has been implicitly accounted for in the Select routine.

```

        MOVE 'SELECT-RUNCTL(SELECT)' TO ERR-SECTION OF SQLRT
        PERFORM ZZ000-SQL-ERROR
    END-IF

    INITIALIZE SELECT-DATA OF S-RUNCTL

    CALL 'PTPSQLRT' USING ACTION-FETCH OF SQLRT
                        SQLRT
                        SQL-CURSOR-COMMON OF SQLRT

    IF RTNCD-ERROR OF SQLRT

        IF RTNCD-END OF SQLRT

            DISPLAY 'Delete Balances Run Control Missing.'
            DISPLAY ' for Operator ID ' OPRID OF S-RUNCTL
            DISPLAY ' and Batch Run ID ' BATCH-RUN-ID OF S-RUNCTL
            SET RTNCD-USER-ERROR OF SQLRT TO TRUE
            PERFORM ZZ000-SQL-ERROR
        ELSE
            MOVE 'SELECT-RUNCTL(FETCH)' TO ERR-SECTION OF SQLRT
            PERFORM ZZ000-SQL-ERROR
        END-IF
    ELSE
        PERFORM DD000-RUNCTL-ACCEPTED
    END-IF

    .
SELECT-RUNCTL-EXIT.

```

KEY POINTS

- 1 Modifying the delivered COBOL process to accept an additional Run Control parameter requires:
 - defining the modification requirements
 - modifying the DMS script to include our new Run Control field in the `Select` statement, and once modified, loading the new Run Control field into the SQL statement table using Data Mover.
 - adding our new Run Control field to both the `Select Setup` and `Select Data` areas.
 - placing logic in the Main section of the COBOL program, which determines which `Purge` routines to execute based on the contents of our new Run Control field.
 - no explicit coding is required in the procedure to select the Run Control parameters. The routine itself requires no modifications due to the methodology used by PeopleSoft.



C H A P T E R 3 4

Additional topics

34.1 Process Scheduler API 749

34.2 Using trace files 758

34.3 Cross reference files 765

Chapters 32 and 33 described the key aspect of PeopleSoft's particular flavor of COBOL: the manner in which the database is accessed. Numerous books have been written about the COBOL language itself. When you purchase the MicroFocus compiler, you receive documentation that may appear to be a small library! While this book does not make any attempt to explain standard COBOL material, we do want to cover some additional topics in COBOL that relate specifically to PeopleSoft.

34.1 PROCESS SCHEDULER API

This section deals with interfacing with the Process Scheduler through PeopleSoft COBOL. When a process is submitted, you can view the job status through Process Monitor. The COBOL program needs to update the run status information on the Process Monitor panel so you will know if the program is processing successfully or if it has encountered an error. To illustrate how the run status information is updated, we'll monitor a COBOL process and point out the code required to update the run status information.

34.1.1 The PTCUSTAT copybook and PTPUSTAT module

The best place to start is with the PTCUSTAT.CBL copybook, which contains the Process Scheduler interface structure. This copybook must exist as an 01-level item in your program.

```
/*****  
*          PROCESS SCHEDULER REQUEST STATUS INTERFACE          *  
*****/  
01  USTAT.                                COPY PTCUSTAT.
```

The following shows an 01-level item called USTAT that includes the PTCUSTAT copybook:

Let's look at the field definitions that comprise the Process Scheduler interface structure. Values may be assigned to the fields in the copybook. This is how the run information is communicated to the Process Monitor:

```
03  PROCESS-INSTANCE      PIC S9(10)  VALUE ZERO  COMP-3.  
03  RUN-STATUS            PIC 9(4)     VALUE ZERO  COMP.  
    88  RUN-STATUS-CANCEL          VALUE 1.  
    88  RUN-STATUS-DELETE          VALUE 2.  
    88  RUN-STATUS-ERROR           VALUE 3.  
    88  RUN-STATUS-HOLD            VALUE 4.  
    88  RUN-STATUS-QUEUED          VALUE 5.  
    88  RUN-STATUS-INITIATED       VALUE 6.  
    88  RUN-STATUS-PROCESSING      VALUE 7.  
    88  RUN-STATUS-CANCELLED       VALUE 8.  
    88  RUN-STATUS-SUCCESSFUL     VALUE 9.  
    88  RUN-STATUS-UNSUCCESSFUL    VALUE 10.  
03  RUN-STATUS-MSGSET      PIC S9(5)   VALUE ZERO  COMP.  
    88  MSGSET-PRCS-SCHED          VALUE 65.  
03  RUN-STATUS-MSGID      PIC S9(5)   VALUE ZERO  COMP.  
    88  MSGID-SUCCESSFUL          VALUE 35.  
    88  MSGID-UNSUCCESSFUL        VALUE 43.  
03  RC                    PIC S9(4)   VALUE ZERO  COMP.  
    88  RC-SUCCESSFUL             VALUE 0.  
03  RC-CHAR               REDEFINES RC  
                           PIC X(2).  
03  MESSAGE-PARM1         PIC X(30)   VALUE SPACE.
```

```

03 MESSAGE-PARM2          PIC X(30)  VALUE SPACE.
03 MESSAGE-PARM3          PIC X(30)  VALUE SPACE.
03 MESSAGE-PARM4          PIC X(30)  VALUE SPACE.
03 MESSAGE-PARM5          PIC X(30)  VALUE SPACE.
03 CONTINUE-JOB           PIC 9(4)    VALUE ZERO  COMP.
    88 CONTINUE-JOB-NO      VALUE 0.
    88 CONTINUE-JOB-YES     VALUE 1.

03 PRUNSTATUS-RC          PIC S9(4)   VALUE ZERO  COMP.
    88 PRUNSTATUS-RC-OK     VALUE ZERO.

03 CALLING-PROGRAM        PIC X(8)    VALUE SPACE.
    88 CALLED-FROM-PTPUPRCS VALUE 'PTPUPRCS'.

```

A call to the delivered module PTPUSTAT performs the actual update:

```

INITIALIZE USTAT
MOVE PROCESS-INSTANCE OF SQLRT TO PROCESS-INSTANCE OF USTAT
SET RUN-STATUS-PROCESSING OF USTAT TO TRUE

CALL 'PTPUSTAT' USING  SQLRT
                      USTAT

```

The sample call displayed sets the run status to 'Processing'. First, the entire interface structure denoted by the 01 group level item USTAT is initialized. The process instance of the current program is then assigned along with the desired run status. The PTPUSTAT module updates the PSPRCSRQST entry for the process instance specified in the PROCESS-INSTANCE field, and Process Monitor displays the information contained in the PSPRCSRQST table.

Let's take a look at the Process Monitor panel after a process has completed.

Figure 34.1 shows the Process Monitor panel after a COBOL process is completed successfully. By double-clicking on the highlighted line, we can examine some additional details about the run.

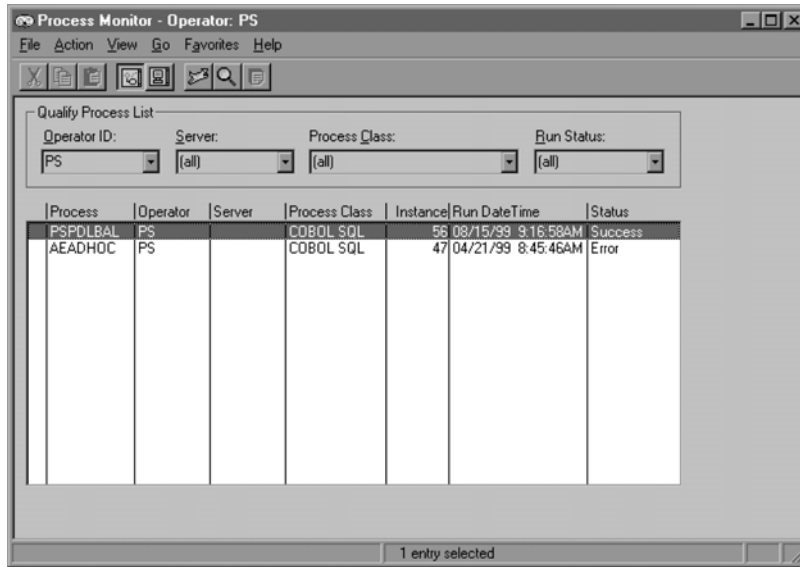


Figure 34.1 The Process Monitor panel

A panel with two folder tabs appears. The first folder tab, called “Process Detail” (figure 34.2), contains information on the run such as operating system, database type, and beginning and ending times.

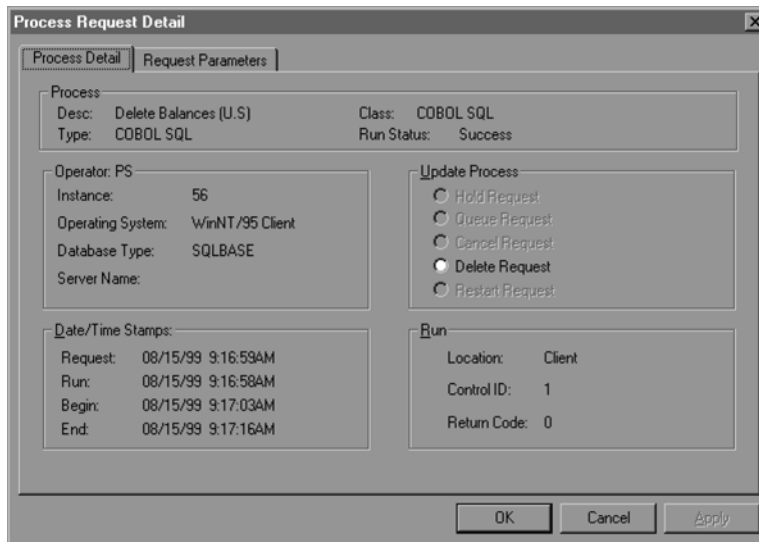


Figure 34.2 The Process Monitor Details folder tab

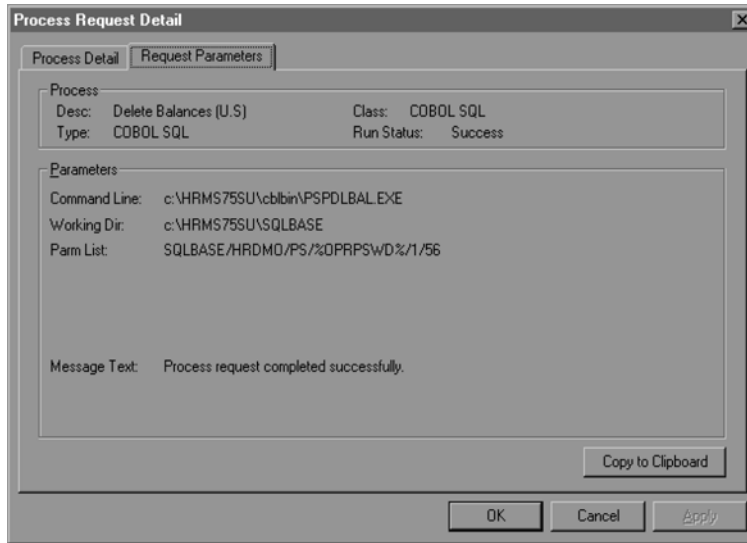


Figure 34.3 The Process Monitor Request Parameters folder tab

The second folder tab, “Request Parameters,” displays additional run information such as the COBOL execution string, the working directory, and the parameters passed to the program.

Take special note of the parameter list passed to the COBOL program. It consists of the database type, the database name, the operator ID, the password, the Run Control ID, and the process instance. If a COBOL process is submitted through the process scheduler, a process instance is assigned. If it is submitted outside of PeopleSoft and the Process Scheduler, the process instance is set to zero. A COBOL process can be executed in its native environment through, for example, the MS-DOS prompt or Unix command line. In that case, the parameters are entered by the user through a series of prompts. The process instance may be entered if you are restarting an aborted COBOL process. Otherwise, the default of zero is used. Before any updates to the Process Monitor are made, the process instance needs to be interrogated to determine if the Process Monitor is being utilized.

```
IF PROCESS-INSTANCE OF SQLRT NOT = ZERO

    PERFORM SD000-SET-RUN-STAT-SUCCESSFUL

END-IF
```

If the process instance does not equal zero, then any updates to the run information may be executed. The preceding code performs the SD000-SET-RUN-STAT-SUCCESSFUL only if a valid process instance is used.

Let's take a look at a real life example. Once again, we'll use the process to delete obsolete payroll balances called PSPDLBAL.

34.1.2 A real life example

We're going to execute the COBOL process to delete obsolete payroll balances. We'll fill in the required Run Control parameters first.

Once the Run Control panel is set with the parameters (figure 34.4), it is then saved. Click on the Traffic Light to initiate the run request.

Manage Payroll Process U.S. - Process - Delete Balances

File Edit View Go Favorites Use Process Inquire Report 1 Report 2 Help

Delete Balances

Operator ID: PS

Run Control ID: 1

Company: CCB Continental Commerce&Business

Balance ID: CY Calendar Year

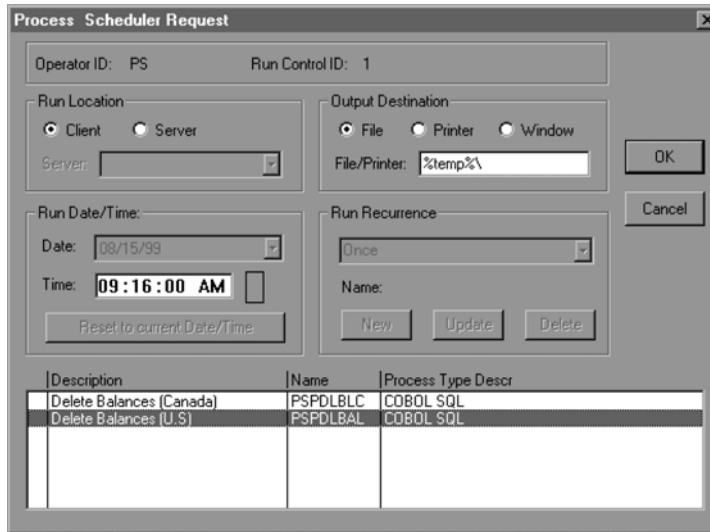
Balance Year: 1992

Period: 1 January

Delete Balances Update

Figure 34.4 Setting the Run Control parameters

The Process Request screen appears (figure 34.5). Select the correct process from the attached list (in our case, the U.S. balance process). If the Run Location and Output Destination are correct, click the OK button to execute the PSPDLBAL process.



Operator ID: PS Run Control ID: 1

Run Location:
☒ Client ☐ Server
 Server:

Output Destination:
☒ File ☐ Printer ☐ Window
 File/Printer: %temp%\

Run Date/Time:
 Date: 08/15/99
 Time: 09:16:00 AM

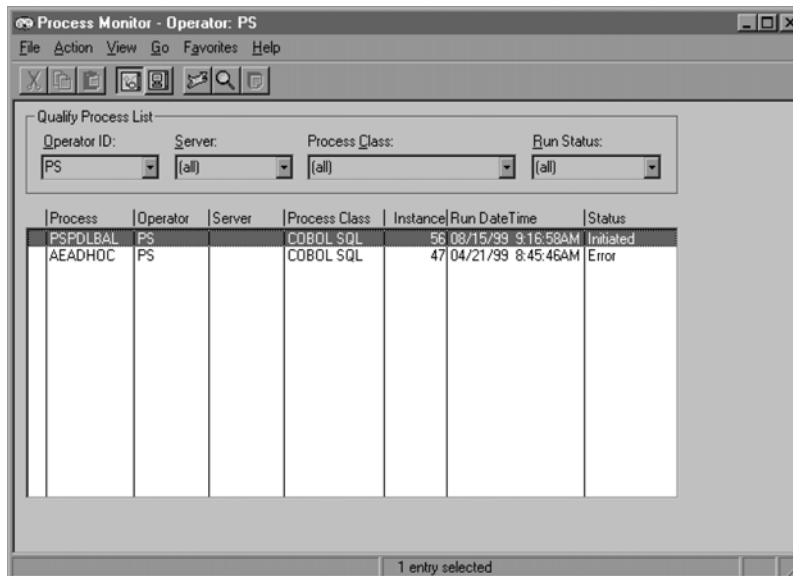
Run Recurrence:

 Name:

Description	Name	Process Type Descr
Delete Balances (Canada)	PSPDLBLC	COBOL SQL
Delete Balances (U.S.)	PSPDLBAL	COBOL SQL

Figure 34.5 Executing a Process Scheduler request

If we were to immediately view the Process Monitor screen, we would see the status is set to 'Initiated' (figure 34.6), the default status before the COBOL program has started. Once the COBOL process takes over, it should set the run status to 'Processing'.



Process Monitor - Operator: PS

File Action View Go Favorites Help

Qualify Process List:
 Operator ID: PS Server: (all) Process Class: (all) Run Status: (all)

Process	Operator	Server	Process Class	Instance	Run DateTime	Status
PSPDLBAL	PS		COBOL SQL	56	08/15/99 9:16:58AM	Initiated
AEADHOC	PS		COBOL SQL	47	04/21/99 8:45:46AM	Error

1 entry selected

Figure 34.6 PSPDLBAL process in Initiated phase

The run status shown in figure 34.7 is now set to Processing. This was controlled by the COBOL process by changing the Run Status field and calling the PTPUSTAT module.

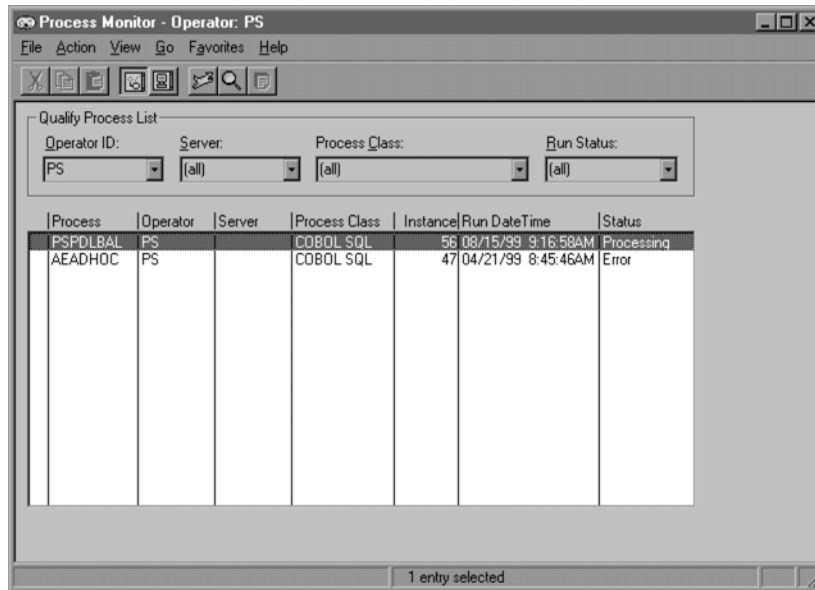


Figure 34.7 PSPDLBAL status moved to Processing

Let's look at a common method of determining whether the process was run through Process Scheduler. If not, an alternate method is used to select the Run Control parameters.

```

IF PROCESS-INSTANCE OF SQLRT NOT = ZERO

    PERFORM DB000-SET-RUN-STAT-PROCESSING
    MOVE PROCESS-INSTANCE OF SQLRT
      TO PROCESS-INSTANCE-ERRMSG OF PSLCT
ELSE

    CALL 'PTPRUNID' USING    SQLRT
                          PROCESS-INSTANCE-ERRMSG
                          OF W-PRC-INSTANCE
    MOVE PROCESS-INSTANCE-ERRMSG OF W-PRC-INSTANCE
      TO PROCESS-INSTANCE-ERRMSG OF PSLCT
END-IF

```

Now, let's look at the steps required to set the run status to Processing. First, the USTAT interface area is initialized. This must be done the first time *only*. The Process

Instance and Run Status fields are assigned. A call to PTPUSTAT is performed followed by the appropriate error handling.

```
INITIALIZE USTAT
MOVE PROCESS-INSTANCE OF SQLRT TO PROCESS-INSTANCE OF USTAT
SET RUN-STATUS-PROCESSING OF USTAT TO TRUE

CALL 'PTPUSTAT' USING SQLRT
                        USTAT
IF RTNCD-ERROR OF SQLRT

    MOVE 'SET-RUN-STAT-PROCESSING(PTPUSTAT) '
        TO ERR-SECTION OF SQLRT
    PERFORM ZZ000-SQL-ERROR
END-IF
```

Figure 34.8 shows the MS-DOS box that appears while the program is running. This may appear for only a few seconds if there isn't much processing to be performed. Any DISPLAY commands in the COBOL program will appear here.



Figure 34.8 PSPDLBAL process executing (MS-DOS Box)

Looking at the Run Status field in figure 34.9, you can see our process ended successfully. Note the process below PSPDLBAL had some problems. The status has been set to Error.

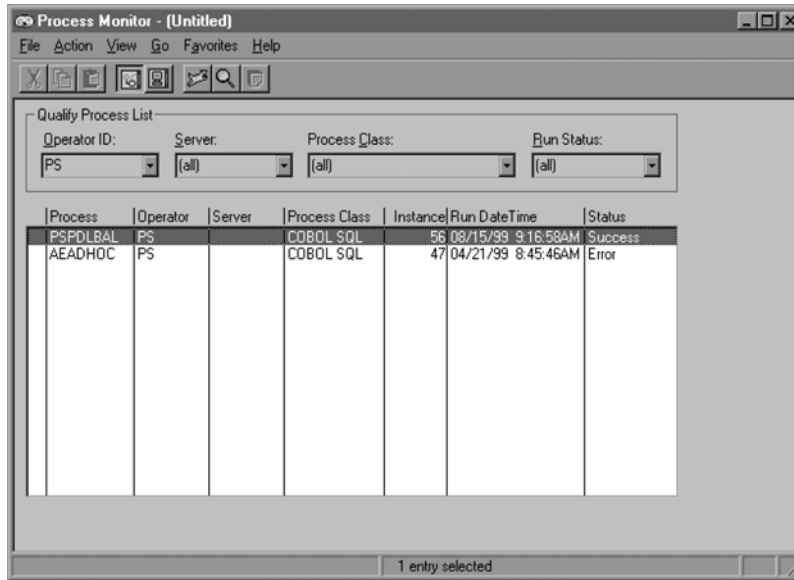


Figure 34.9 Viewing the results using Process Monitor

The following code sets the run status to 'Success'. First the new Run Status field is assigned, then the PTPUSTAT module performs the update.

```

SET RUN-STATUS-SUCCESSFUL OF USTAT TO TRUE

CALL 'PTPUSTAT' USING   SQLRT
                        USTAT
IF RTNCD-ERROR OF SQLRT

    MOVE 'SET-RUN-STAT-SUCCESSFUL (PTPUSTAT) '
        TO ERR-SECTION OF SQLRT
    PERFORM ZZ000-SQL-ERROR
END-IF

```

The following code sets the run status to 'Error'. Once again, the new run status is set and the call to PTPUSTAT performs the update.

```

SET RUN-STATUS-ERROR OF USTAT TO TRUE

CALL 'PTPUSTAT' USING   SQLRT
                        USTAT
IF RTNCD-ERROR OF SQLRT

    MOVE 'SET-RUN-STAT-ERROR (PTPUSTAT) '
        TO ERR-SECTION OF SQLRT
    PERFORM ZZ000-SQL-ERROR
END-IF

```

34.2 USING TRACE FILES

Ultimately, the goal of updating the Process Status is to enable the user to monitor the progress of their application. This is a very high-level method of monitoring. To actually determine what is happening within the program, a trace file may be generated. This is used primarily by the technical staff to debug problems or perform performance-tuning functions. Some highly skilled functional or “power” users may also use trace files to become more familiar with processes.

Configuration Manager is a PeopleTool that allows you to update PeopleSoft registry entries. Updates are made through a GUI interface that is much easier to use than the Windows Registry editor (regedit). A series of folder tabs groups each set of entries by category or function. Some of these include Startup, Process Scheduler, and Client Setup parameters. Trace settings may be turned on using Configuration Manager.

Click on the Trace folder tab to designate the desired tracing level. Three sections exist within the Trace panel: One section controls trace settings for PeopleCode; another controls the Message Agent trace; the third group of Trace settings, under the label “SQL Trace,” controls the Trace settings for online activity and COBOL processes. Online activity is defined as all SQL activity generated by the Panel Processor. We’re interested in the COBOL Trace settings. The checkboxes indicate the Trace level combinations you would like to use.

You can see the Configuration Manager trace settings in figure 34.10. Notice the Online Trace File edit box. Normally, this is used to redirect your trace file output. COBOL trace file output cannot be redirected using this field.

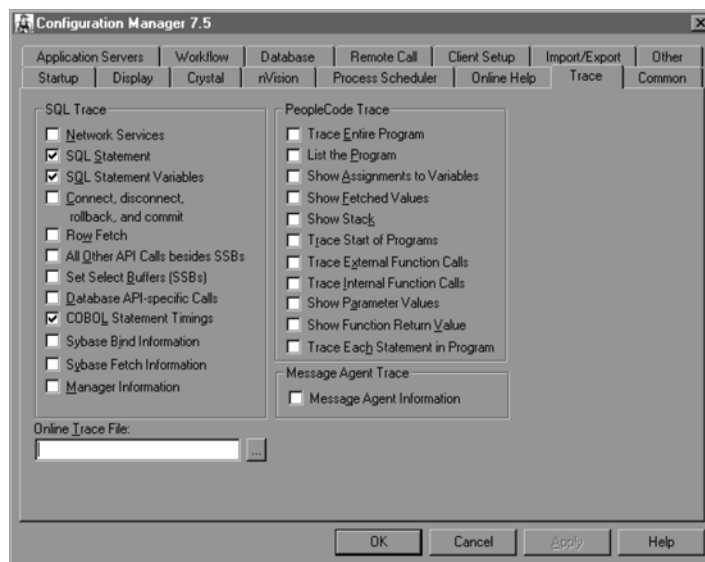


Figure 34.10 Configuration Manager trace settings

The COBOL trace file is written to the following directory:

For Windows: %TEMP%\ps\<database_name>

For Unix: \$PS_HOME/log/<database_name>

The filename has the following format:

COBSQL[_progrname][_processinstance | _MMDDHHMMSS}.TRC

Depending on the COBOL process, the trace filename may vary. Special coding exists in each program that dictates the trace filename to use. The sample COBOL process we've been using (PSPDLBAL) writes the trace file as COBSQL_<process_instance>.TRC when run through Process Scheduler. When it is executed outside of PeopleSoft (MS-DOS prompt), it uses the format COBSQL_<MMDDHHMMSS>.TRC since no process instance is provided.

34.2.1 Trace settings

Each trace setting checkbox has a unique value. The combination of selected trace setting values are added and stored in the windows registry via Configuration Manager. Let's take a closer look at the trace values and how they are stored.

Table 34.1 SQL Trace values

Trace function	Decimal value	Binary value
SQL statement	1	0000 0000 0000 0001
SQL statement Variables	2	0000 0000 0000 0010
Connect, Disconnect, Rollback and Commit	4	0000 0000 0000 0100
Row Fetch	8	0000 0000 0000 1000
All other API calls besides SSBs	16	0000 0000 0001 0000
Set Select Buffers (SSBs)	32	0000 0000 0010 0000
Database API-specific calls	64	0000 0000 0100 0000
COBOL statement timings	128	0000 0000 1000 0000
Sybase Bind information	256	0000 0001 0000 0000
Sybase Fetch information	512	0000 0010 0000 0000
N/A	1024	0000 0100 0000 0000
Network services	2048	0000 1000 0000 0000
Manager information	4096	0001 0000 0000 0000

The individual trace setting values are shown in table 34.1. For each trace level, the decimal value and binary equivalent are presented. When your trace settings are applied, the values are added and stored in the Windows Registry.

The registry address (or key) is

My Computer\HKEY_CURRENT_USER\Software\PeopleSoft\
PeopleTools\Release7.5\Trace

The registry field (or subkey) is:

TraceSQL

Let's take a look at the Windows Registry after we apply the trace settings used in figure 34.10. In this particular example, we have selected the checkboxes to turn on the SQL statement, SQL statement variable, and COBOL statement timing levels. The corresponding values for these are 1, 2, and 128. The TraceSQL registry field (also referred to as a subkey) contains the total value of the selected checkboxes, which is 131 (1 + 2 + 128).

Using the Windows Registry Editor (regedit.exe), we can open the PeopleSoft Configuration Manager key and look at the TraceSQL subkey contents. The Registry Editor can be found in the Windows directory. Notice the key value in the Registry Editor window on the bottom of the screen. This is the full path key that leads to the TraceSQL subkey. You can see it contains the total of the selected trace values (131). The data in the subkey is displayed as a hexadecimal value (x'83') with the decimal equivalent in parentheses. Take special caution when using the Windows Registry editor. Incorrectly set values or transferred subkeys may cause serious problems with the Windows operating system.

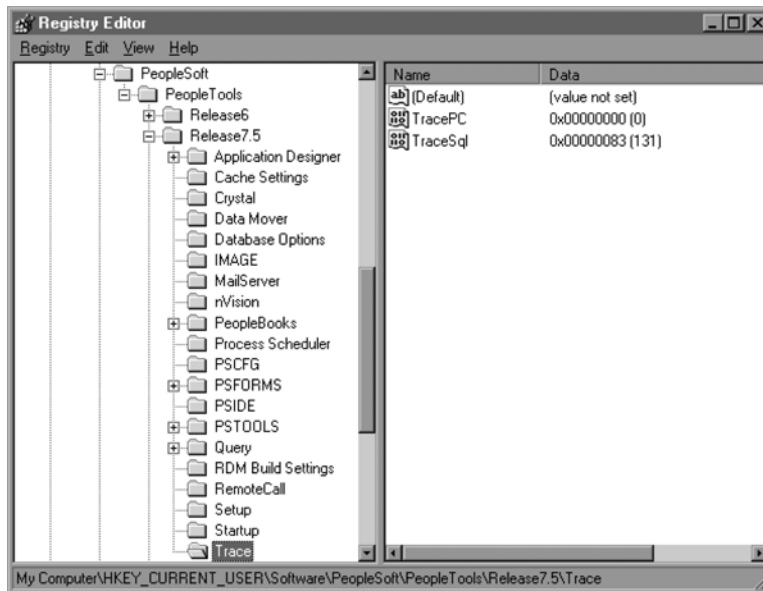


Figure 34.11 PeopleSoft registry entries

34.2.2 Tracing a COBOL process

Now let's see the COBOL Trace in action. We'll trace the PSPDLBAL process we've been using throughout this chapter (Delete Obsolete Balances). We'll use the same trace values as shown in figure 34.10 (SQL statement, SQL statement variable, and COBOL statement timings). Since we've already set these, we can move directly to the Run Control panel and initiate the process.

Once we fill in the parameters for our run (figure 34.12), we click on the Traffic Light to initiate the process. The COBOL process detects that the trace levels have been set through the Windows Registry subkey `TraceSQL` and produces the trace file in the default directory.

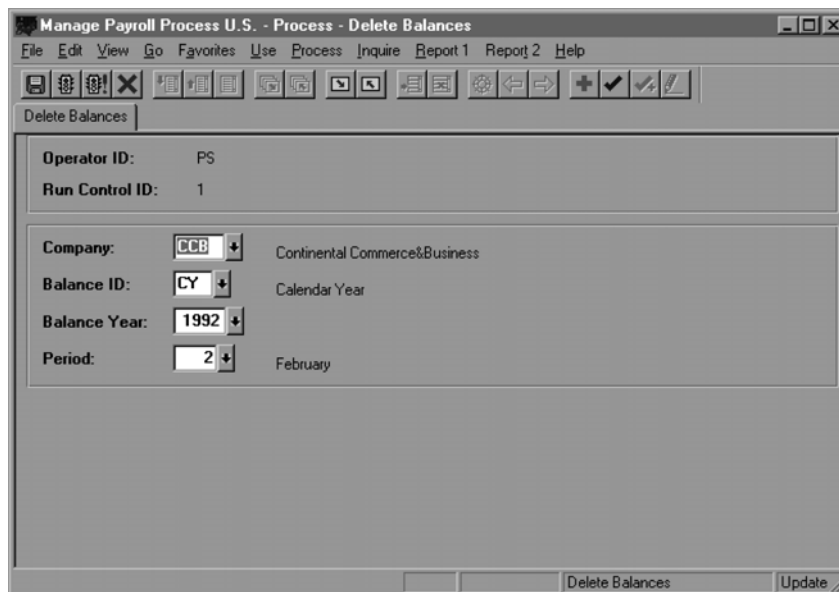


Figure 34.12 Setting the Run Control parameters

Figure 34.13 displays the results on the Process Monitor. Our run was successful. The process instance for our run was 61. Let's take a look at the trace file generated during this run.

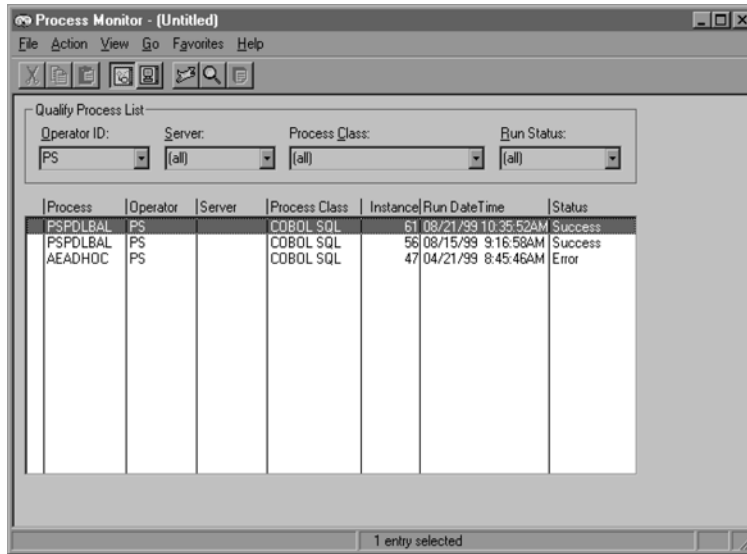


Figure 34.13 The process PSPDLBAL has ended successfully

Figure 34.14 shows the COBOL trace file as it appears in Windows Explorer. The directory is `C:\WINDOWS\TEMP\PS\HRDMO`, and the trace filename is `COBSQL_61.TRC`. This adheres to the naming conventions we spoke of earlier.

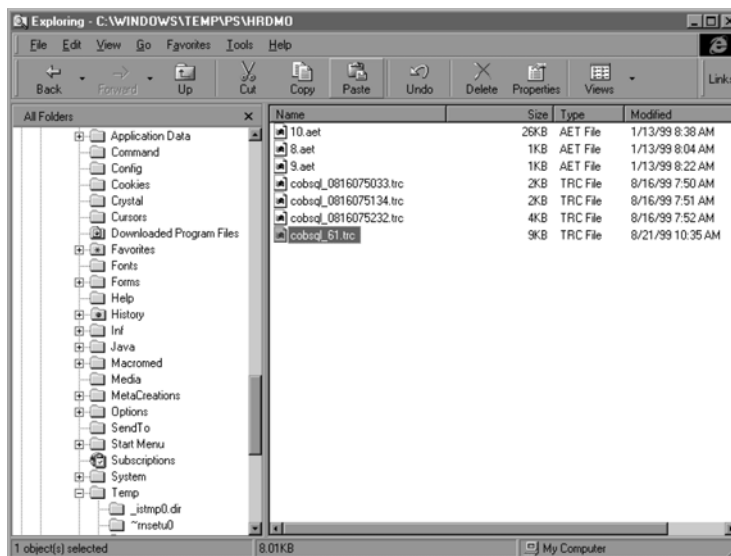
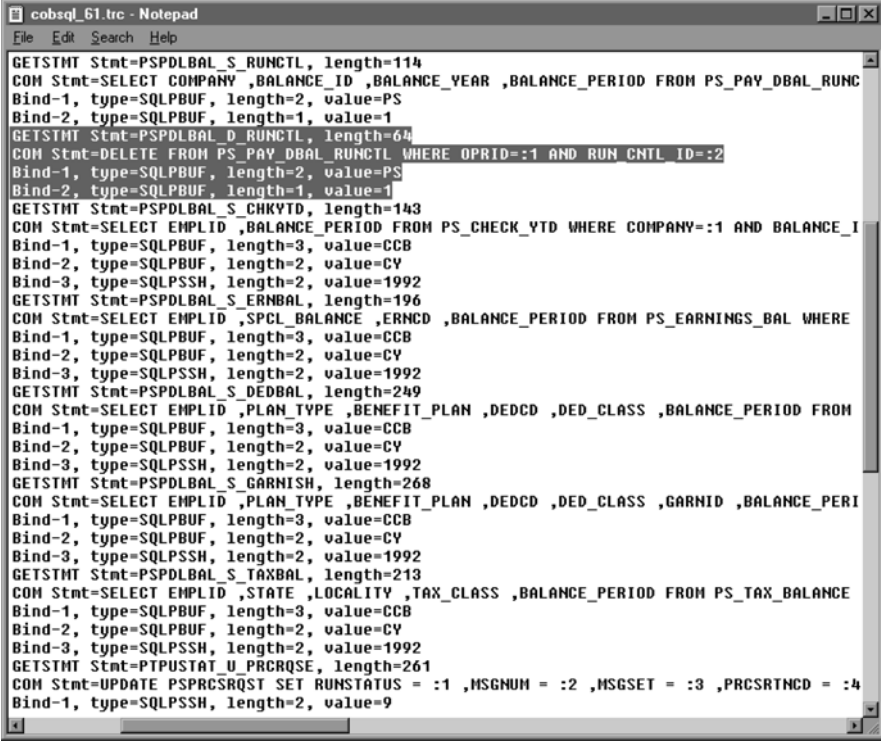


Figure 34.14 Locating the COBOL trace file

34.2.3 Examining the trace file contents

We can view the trace file contents using an editor or word processor. Let's take a look at the trace output now.

Since our trace file is relatively small, we can open and view it using Notepad (figure 34.15). Take a look at the highlighted area. The first highlighted line shows the word `GETSTMT` followed by the stored SQL statement retrieved by the `PTPSQLRT` module. The name of it is `PSPDLBAL_D_RUNCTL`. Once retrieved, the next line displays the SQL statement text to be compiled within the COBOL process. Notice the `:1` and `:2` bind variables used in the criteria of the SQL statement. The third and fourth highlighted lines show the data used when resolving the `:1` and `:2` bind variables. If you recall from earlier chapters, the bind variables are defined within the Bind Setup and Bind Data areas for each stored SQL statement called. The `:1` bind variable is used for the `OPRID` which is set to the value 'PS'. The `:2` bind variable is used for the `RUN_CNTL_ID` and has been set to 1. You can verify these values by looking at the Run Control panel in figure 34.12. All other stored SQL statements and the SQL statement variables appear in the trace file. This is because we have specifically checked them using Configuration Manager.

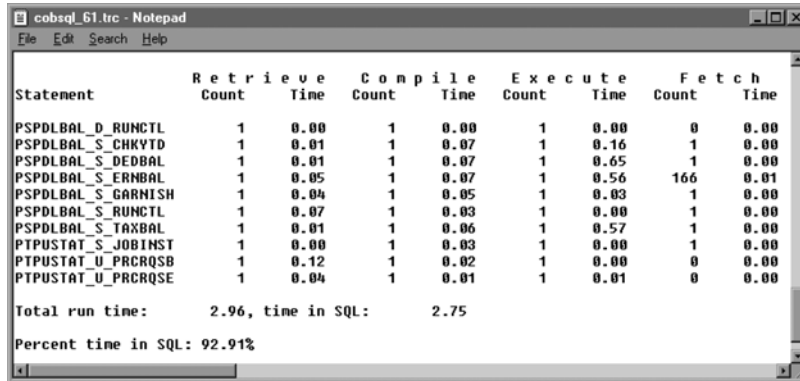


```
cobsql_61.trc - Notepad
File Edit Search Help

GETSTMT Stmt=PSPDLBAL_S_RUNCTL, length=114
COM Stmt=SELECT COMPANY ,BALANCE_ID ,BALANCE_YEAR ,BALANCE_PERIOD FROM PS_PAY_DBAL_RUNCT
Bind-1, type=SQLPBUF, length=2, value=PS
Bind-2, type=SQLPBUF, length=1, value=1
GETSTMT Stmt=PSPDLBAL_D_RUNCTL, length=64
COM Stmt=DELETE FROM PS_PAY_DBAL_RUNCTL WHERE OPRID=:1 AND RUN_CNTL_ID=:2
Bind-1, type=SQLPBUF, length=2, value=PS
Bind-2, type=SQLPBUF, length=1, value=1
GETSTMT Stmt=PSPDLBAL_S_CHKVTD, length=143
COM Stmt=SELECT EMPLID ,BALANCE_PERIOD FROM PS_CHECK_YTD WHERE COMPANY=:1 AND BALANCE_I
Bind-1, type=SQLPBUF, length=3, value=CCB
Bind-2, type=SQLPBUF, length=2, value=CY
Bind-3, type=SQLPSSH, length=2, value=1992
GETSTMT Stmt=PSPDLBAL_S_ERNBAL, length=196
COM Stmt=SELECT EMPLID ,SPCL_BALANCE ,ERNCD ,BALANCE_PERIOD FROM PS_EARNINGS_BAL WHERE
Bind-1, type=SQLPBUF, length=3, value=CCB
Bind-2, type=SQLPBUF, length=2, value=CY
Bind-3, type=SQLPSSH, length=2, value=1992
GETSTMT Stmt=PSPDLBAL_S_DEDBAL, length=249
COM Stmt=SELECT EMPLID ,PLAN_TYPE ,BENEFIT_PLAN ,DEDCD ,DED_CLASS ,BALANCE_PERIOD FROM
Bind-1, type=SQLPBUF, length=3, value=CCB
Bind-2, type=SQLPBUF, length=2, value=CY
Bind-3, type=SQLPSSH, length=2, value=1992
GETSTMT Stmt=PSPDLBAL_S_GARNISH, length=268
COM Stmt=SELECT EMPLID ,PLAN_TYPE ,BENEFIT_PLAN ,DEDCD ,DED_CLASS ,GARNID ,BALANCE_PERI
Bind-1, type=SQLPBUF, length=3, value=CCB
Bind-2, type=SQLPBUF, length=2, value=CY
Bind-3, type=SQLPSSH, length=2, value=1992
GETSTMT Stmt=PSPDLBAL_S_TAXBAL, length=213
COM Stmt=SELECT EMPLID ,STATE ,LOCALITY ,TAX_CLASS ,BALANCE_PERIOD FROM PS_TAX_BALANCE
Bind-1, type=SQLPBUF, length=3, value=CCB
Bind-2, type=SQLPBUF, length=2, value=CY
Bind-3, type=SQLPSSH, length=2, value=1992
GETSTMT Stmt=PTPSTAT U_PCRQSE, length=261
COM Stmt=UPDATE PTPCRSQST SET RUNSTATUS = :1 ,MSGNUM = :2 ,MSGSET = :3 ,PCRSRTNCD = :4
Bind-1, type=SQLPSSH, length=2, value=9
```

Figure 34.15 Viewing a portion of the trace file

We also selected the COBOL statement timings checkbox. At the end of the trace file, we can see the COBOL statement timings (figure 34.16). The checkbox shows statistics for each stored SQL statement processed by the COBOL program. This can be used for performance tuning functions.



Statement	Retrieve Count	Retrieve Time	Compile Count	Compile Time	Execute Count	Execute Time	Fetch Count	Fetch Time
PSPDLBAL_D_RUNCTL	1	0.00	1	0.00	1	0.00	0	0.00
PSPDLBAL_S_CHKYTD	1	0.01	1	0.07	1	0.16	1	0.00
PSPDLBAL_S_DEDBAL	1	0.01	1	0.07	1	0.65	1	0.00
PSPDLBAL_S_ERNBAL	1	0.05	1	0.07	1	0.56	166	0.01
PSPDLBAL_S_GARNISH	1	0.04	1	0.05	1	0.03	1	0.00
PSPDLBAL_S_RUNCTL	1	0.07	1	0.03	1	0.00	1	0.00
PSPDLBAL_S_TAXBAL	1	0.01	1	0.06	1	0.57	1	0.00
PTPUSTAT_S_JOBINST	1	0.00	1	0.03	1	0.00	1	0.00
PTPUSTAT_U_PRCRQSB	1	0.12	1	0.02	1	0.00	0	0.00
PTPUSTAT_U_PRCRQSE	1	0.04	1	0.01	1	0.01	0	0.00
Total run time: 2.96, time in SQL: 2.75								
Percent time in SQL: 92.91%								

Figure 34.16 Viewing the COBOL statement timings

NOTE Once you have produced the trace output, make certain you turn off the trace options. If you don't, all subsequent activity will continue to be traced, which can greatly affect performance.

A simple batch program can be written to automatically deactivate the trace options when you restart Windows. Simply place the batch program in the Startup directory on the workstation. Here is an example, using a language called WinBatch (You can find information about WinBatch at <http://www.windowware.com>):

```
ErrorMode(@OFF)

Rpath = "Software\PeopleSoft\PeopleTools\Release75\Trace"
Rkey = RegOpenKey(@REGCURRENT, Rpath)
Rerr = LastError()

if Rerr == 0

    TraceFile = ''
    TracePC = 0
    TraceSQL = 0

    RegSetValue(Rkey, "[TraceFile]", TraceFile)
```

```

    RegSetDword(Rkey, "[TraceSQL]", TraceSQL)
    RegSetDword(Rkey, "[TracePC]", TracePC)

    RegCloseKey(Rkey)

endif

ErrorMode(@CANCEL)

```

The preceding sample WinBatch script simply opens the key node of the registry and updates the subkeys with NULL or zeroes. The NULL is used for the `TraceFile` output subkey, and the zeroes will override the SQL Trace (Online/COBOL) and PeopleCode Trace subkey settings. Finally, the key node is closed. When the user signs on to PeopleSoft, the trace values will no longer be set.

34.3 CROSS REFERENCE FILES

PeopleSoft provides Cross-Reference report files, which accompany the delivered COBOL processes and help explain how they work. Some .xrf files list the programs and the modules they call. Others list the stored SQL statements or the database tables referenced by the SQL statements.

An example of a delivered Cross-Reference report is found in figure 34.17. It lists the COBOL processes and a tree listing of the modules called. Notice the highlighted area, which shows the sample COBOL process we have been using (PSPDLBAL). You can see that the PSPDLBAL module called three other modules: PTPRUNID, PTPSQLRT, and PTPUSTAT.

All of the delivered Cross-Reference reports are static. If any COBOL process is modified, they cannot be reproduced to reflect the latest version. The Cross-Reference listings may need to be tracked and updated manually (assuming it is being used as actual documentation for all updates).

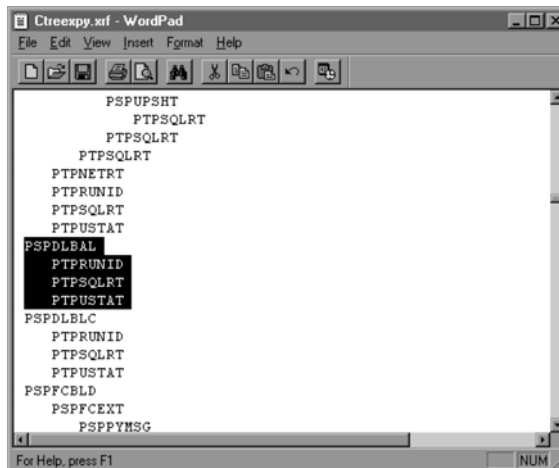


Figure 34.17
Delivered Call Structure
Cross-Reference report

In part 7, “Using Application Engine,” we will discuss an SQR utility that can be downloaded and used to flowchart Application Engine programs. The same site contains utilities that produce an updated COBOL Call Structure listing (similar to the static Cross-Reference report) and a utility that flowcharts the actual COBOL perform structure (COBOL Analyzer). Both can be found on the site:

<http://www.sqrtools.com>.

The COBOL Analyzer produces a nested process flowchart of all the performed sections within a program. This can be extremely useful when looking through some of the larger processes such as those in PeopleSoft Payroll.

Let’s look as a sample of the COBOL Analyzer output listing:

```
Report ID:  TDCBL02                COBOL ANALYZER                Page No.   6
                                           Run Date 06/12/1999
Program:   c:\src\picmpar.cbl                Run Time 10:26:34
=====
Process Flowchart
=====

AA000-MAIN
  XA000-LOAD-FILE-DEFN
    <CALL>.PIPUTLTY
    XA100-SET-STOP-DATE
  XW000-GET-START-TIME
    XZ000-GET-CLOCK-TIME
  DA000-BUILD-PARTIC-LIST
    DD000-INSERT-PIPRT
      DE000-SELECT-PARTIC-CURRENT
        ZM000-MESSAGE
          <CALL>.PSPPYMSG
        DF000-FETCH-PARTIC-CURRENT
          ZM000-MESSAGE <R>
        DG000-SELECT-PARTIC-PRIOR
          ZM000-MESSAGE <R>
        DH000-FETCH-PARTIC-PRIOR
          ZM000-MESSAGE <R>
        DI000-INSERT-PARTIC-DATA
          ZM000-MESSAGE <R>
        DF000-FETCH-PARTIC-CURRENT <R>
```

The first section above is AA000-MAIN. All subsequent sections appear in execution order. If the program is modified, either through customizations or upgrade patches, the analyzer can be run again and a new listing generated.

KEY POINTS

- 1** PeopleSoft COBOL programs interface with Process Scheduler using the PTPUSTAT module and the PTCUSTAT copybook. Specific COBOL code is called to update Process Monitor fields.
- 2** Set the SQL Trace levels using Configuration Manager. The trace values are stored in the Windows Registry. Make sure you turn the trace off when you're done. Since the online panels use the same trace settings as COBOL, system performance can be greatly affected.
- 3** You can use the trace to view the SQL statements executed along with the resolved bind variable contents. You may use several other trace options when troubleshooting.
- 4** PeopleSoft delivers Cross-Reference reports for their COBOL processes. You can also find additional downloadable utilities for COBOL processes at <http://www.sqrtools.com>.

Using Application Engine

Application Engine (A/E), a PeopleTool introduced in version 5.0, offers an alternative to conventional structured programming. Application Engine programs are created using a series of online panels which allow you to define your application along with any section, step, and statement components. You can use radio buttons, checkboxes, and drop-down lists to designate execution options in your program. SQL statements are entered on the statement panel in an edit box. All of this information is saved within the database and utilized by the Application Engine driver program PTPMAIN. Because Application Engine is not an intuitive development tool, we follow our introduction to Application Engine with a “hands-on” approach, presenting a series of exercises in a tutorial designed to illuminate the differences between A/E and conventional structured programming. As a prerequisite to part 7, the user should be well-versed in PeopleTools, particularly Application Designer. A good understanding of SQR programming (as well as SQL) would also be helpful. We end the section with an overview of new Application Engine features in release 8.0.



CHAPTER 35

What is Application Engine?

35.1 About Application Engine	771	35.5 A/E definition tables	777
35.2 Advantages/disadvantages	772	35.6 A/E definition panels	779
35.3 Set processing concepts	772	35.7 A/E section/step relationship	786
35.4 The main components of Application Engine	776	35.8 Application Engine: the big picture	788

35.1 ABOUT APPLICATION ENGINE

Application Engine (A/E) is a PeopleTool that allows you to create and execute Batch SQL programs. SQL statements are entered online and processed by the PeopleSoft COBOL program PTPEMAIN. Applications can be broken into smaller pieces called Sections and within these Sections are Steps. Each Step either executes SQL statements, another Section, a COBOL program, or a Mass Change program. In structured programming languages such as SQR, variables are used to store information throughout the life of the program. In Application Engine, a Cache record is used to store values so they may be utilized by subsequent steps in the AE program. As your program proceeds through its Sections/Steps, messages may be written, which are stored in the Message Log tables. These messages may be viewed through the Application Engine Messages panel.

35.2 ADVANTAGES/DISADVANTAGES

Application Engine offers both advantages and disadvantages:

35.2.1 Advantages

- All Application Engine components reside within the database itself. All application development and testing is done within PeopleTools.
- Application Engine programs are considered multi-platform. Database-specific sections can be utilized using a database platform directive that matches your particular installation.
- PeopleSoft Meta-SQL is supported within Application Engine.
- Changes to the PeopleSoft data dictionary are global. No modifications to Application Engine programs are normally required when a field attribute is changed.
- Application Engine programs use effective-dating for each section (or procedure). A history of modifications can be easily maintained instead of overlaid.
- Extremely efficient programs can be created using set processing techniques.

35.2.2 Disadvantages

- Application Engine panels are not intuitive: It can be confusing scrolling through a maze of checkboxes, radio buttons, and folder tabs.
- It is difficult to visualize the flow of an Application Engine program. Sections are stored and displayed in alphabetical order in the list box instead of a more logical order.
- Even the simplest of modifications to Application Engine programs can be a harrowing experience. Some more complex programs need to use temporary tables to pass information from one step to another. The dependencies on these temporary tables by other sections need to be carefully analyzed.

35.3 SET PROCESSING CONCEPTS

The most efficient Application Engine programs use set processing techniques whenever possible. In fact, Application Engine was designed with this technique in mind. Large groups of data with the same criteria can be processed at once instead of individually (or row-by-row). Depending on the volume of data processed, set processing can dramatically improve the overall performance of your program.

The set processing SQL concept has been around for many years. It is used extensively when updating the database using native SQL tools such as SQL*Plus or SQL*Talk. Set processing can also be referred to as a mass update. These mass updates may be split into several SQL statements to accommodate different sets of update criteria for each group of data. Let's look at a simplified example of set processing before we move on to Application Engine Basics. We'll use basic SQR routines to demonstrate set processing in comparison to row by row processing.

35.3.1 Set processing vs. row by row processing

For our example, let's assume we have a record called MY_TABLE. Many fields exist in the table including MY_KEY, DEPTID, and ACCT_TYPE. MY_KEY will serve as the unique table key. Also included is a field called BUSINESS_UNIT, which is not populated. Based on the ACCT_TYPE value, we need to perform two different methods of deriving the BUSINESS_UNIT using the DEPTID field.

If the ACCT_TYPE has a value of 'A', BUSINESS_UNIT will be extracted from a table called MY_CONV_A. If ACCT_TYPE has a value of 'B', the BUSINESS_UNIT will be extracted from a table called MY_CONV_B.

35.3.2 Example of row by row processing

First, we'll use the row-by-row processing technique to derive the business unit:

```
...

begin-select

u.my_key
u.deptid
u.acct_type

    let $NEW_business_unit = ''

    if &u.acct_type = 'A'
        do Select-Conv-A
    else
        do Select-Conv-B
    end-if

    if not isnull($NEW_business_unit)
        do Update-My-Table
    end-if

from ps_my_table
where u.acct_type in ('A','B')

end-select

...
```

The main Select, as indicated, fetches a row from MY_TABLE one by one. Only rows with ACCT_TYPE of 'A' or 'B' are selected. If ACCT_TYPE is equal to 'A', then the Select-Conv-A routine is performed. If ACCT_TYPE is not equal to 'A', then the Select-Conv-B routine is performed by default. If a BUSINESS_UNIT is found in either of these tables, then the routine Update-My-Table is performed. This process will continue until all rows with ACCT_TYPE equal to 'A' or 'B' have been processed.

Let's look at the `Select-Conv-A` routine. If there is a matching entry in the `MY_CONV_A` table for the `DEPTID`, then the `$NEW_business_unit` variable will be set to the `BUSINESS_UNIT` value in the table:

```
begin-procedure Select-Conv-A

begin-select

a.business_unit

    let $NEW_business_unit = &a.business_unit

    from ps_my_conv_a    a
where a.deptid          = &u.deptid

end-select

end-procedure
```

Let's look at the `Select-Conv-B` routine. If there is a matching entry in the `MY_CONV_B` table for the `DEPTID`, then the `$NEW_business_unit` variable will be set to the `BUSINESS_UNIT` value in the table:

```
begin-procedure Select-Conv-B

begin-select

b.business_unit

    let $NEW_business_unit = &b.business_unit

    from ps_my_conv_b    b
where b.deptid          = &u.deptid

end-select

end-procedure
```

Finally, let's have a look at the routine that updates `MY_TABLE` with the new `BUSINESS_UNIT` value (if a matching entry were found):

```
begin-procedure Update-My-Table

begin-sql

update ps_my_table
    set business_unit = $NEW_business_unit
where my_key          = &u.my_key

end-sql

end-procedure
```

Depending on the volume of data that will be processed, the row by row approach may be fine. At a minimum, each row selected from MY_TABLE must perform a Select to retrieve the Business Unit. If it exists, an Update statement is performed. Imagine if the number of rows affected by this process were over 100,000. Maybe even 500,000 or more! This means the Select against a conversion table would be executed that many times. The Update routine could potentially execute the same amount of times! That's a lot of database activity that could be avoided! Network traffic, which could yield the greatest degradation in performance, needs to be considered.

We can implement optimization techniques in our row by row processing example to improve performance. For example, we can order the main Select by ACCT_TYPE and DEPTID. We then perform a Conversion Lookup only when a change to one of these fields occurs. We store and utilize the results based on the changing combination of these two fields in our update routine. Even with these improvements, performance can still be poor when processing large amounts of data one row at a time.

35.3.3 Example of set processing

Our set processing example is much simpler and much more efficient. The improvement in performance increases with the volume of transactions processed. Let's perform the set processing Update using the MY_CONV_A table:

```
...

begin-sql

update ps_my_table          u
  set u.business_unit      =
      (select z.business_unit
       from ps_my_conv_a    z
       where z.deptid       = u.deptid)
where u.acct_type          = 'A'
  and exists
      (select 'X'
       from ps_my_conv_a    x
       where x.deptid       = u.deptid);

commit;

end-sql

...
```

The WHERE clause limits the Update to all rows in MY_TABLE that have ACCT_TYPE equal to 'A' and an existing entry in the MY_CONV_A table equal to the DEPTID on the row. This one Update statement populates all the rows that

match this criteria at once. Now let's perform the set processing Update using the MY_CONV_B table:

```
...  
  
begin-sql  
  
update ps_my_table          u  
  set u.business_unit      =  
      (select z.business_unit  
        from ps_my_conv_b    z  
        where z.deptid       = u.deptid)  
where u.acct_type           = 'B'  
and exists  
  (select 'X'  
    from ps_my_conv_b        x  
    where x.deptid           = u.deptid);  
  
commit;  
  
end-sql  
  
...
```

The WHERE clause limits the Update to all rows in MY_TABLE that have an ACCT_TYPE equal to 'B' and an existing entry in the MY_CONV_B table equal to the DEPTID on the row. This one update statement also populates all the rows that match this particular criteria at once.

TIP It's a good idea to COMMIT frequently during set processing operations. Large amounts of data may be updated at once, causing more system resources to be utilized.

The set processing examples executed two SQL Updates. No further database activity was required, and there was no network traffic at all. The Updates were entirely at the database level.

Always keep set processing in mind when you're developing with Application Engine, and try to use it whenever possible to achieve the maximum performance in your programs.

35.4 THE MAIN COMPONENTS OF APPLICATION ENGINE

Application Engine contains the following main components:

APPLICATION The highest level of an Application Engine program comprised of one or more sections.

SECTION Equivalent to an SQR procedure or COBOL paragraph comprised of one or more steps. An Application Engine program always begins with a section called MAIN.

STEP Can be considered the actual work component of an Application Engine program. In most cases, it is used to execute an SQL statement or call another section. It can also call a COBOL program or a Mass Change program.

STATEMENTS An SQL statement attached to a step. Several statement types are used to qualify a statement: Select, Update/Insert/Delete, DO Select, DO When, DO Until, DO While, and Comment. The Update/Insert/Delete statement type is used not only to update the database but also to insert messages into the message log.

35.5 A/E DEFINITION TABLES

All Application Engine development is done within the database itself. Just as a record or panel definition is created and stored in the database, the same can be said of Application Engine. Using the Application Engine panels, the application is defined and stored in an application table. Next a section is defined and stored in the section table. The same occurs for each step and each statement. Four definition tables are used to store Application Engine programs along with the relationship to one another (table 35.1 through table 35.5).

Table 35.1

AE_APPL_TBL	AE_SECTION_TBL	AE_STEP_TBL	AE_STMT_TBL
AE_PRODUCT	AE_PRODUCT	AE_PRODUCT	AE_PRODUCT
AE_APPL_ID	AE_APPL_ID	AE_APPL_ID	AE_APPL_ID
	AE_SECTION	AE_SECTION	AE_SECTION
	DB_PLATFORM	DB_PLATFORM	DB_PLATFORM
	EFFDT	EFFDT	EFFDT
		AE_STEP	AE_STEP
			AE_STMT_TYPE

We will now briefly describe some of the more important fields stored in each of these tables.

Table 35.2

AE_APPL_TBL	Application Definition Table
AE_PRODUCT	Product
AE_APPL_ID	Application Name
AE_VERSION	Version Number

Table 35.2 (continued)

AE_APPL_TBL	Application Definition Table
DESCR	Description
AE_CACHE_RECNAME	Cache Record Name
MESSAGE_SET_NBR	Message Set Number
AE_DEBUG_MODE	Debug Application
AE_TRACE	Trace Application Steps

Table 35.3

AE_SECTION_TBL	Section Definition Table
AE_PRODUCT	Product
AE_APPL_ID	Application Name
AE_SECTION	Section
DB_PLATFORM	Database Platform
EFFDT	Effective Date
EFF_STATUS	Effective Status
DESCR	Description

Table 35.4

AE_STEP_TBL	Step Definition Table
AE_PRODUCT	Product
AE_APPL_ID	Application Name
AE_SECTION	Section
DB_PLATFORM	Database Platform
EFFDT	Effective Date
AE_STEP	Step Name
AE_SEQ_NUM	Step Sequence Number
EFF_STATUS	Effective Status
PROGRAM_NAME	COBOL Program
MC_DEFN_ID	Mass Change Definition
AE_DO_PRODUCT	DO Product
AE_DO_APPLID	DO Application
AE_DO_SECTION	DO Section
AE_SQL_UPDATE	Edit SQL
AE_SQL_SELECT	Select Present
AE_DO_WHEN	When
AE_DO_WHILE	While
AE_DO_UNTIL	Until

Table 35.4 (continued)

AE_STEP_TBL	Step Definition Table
AE_DO_SELECT	Select
AE_SELECT_END_DO	Select Ends the DO
AE_DO_SELECT_TYPE	Type of DO Select

Table 35.5

AE_STMT_TBL	Statement Definition Table
AE_PRODUCT	Product
AE_APPL_ID	Application Name
AE_SECTION	Section
DB_PLATFORM	Database Platform
EFFDT	Effective Date
AE_STEP	Step Name
AE_STMT_TYPE	Statement Type
AE_STMT	SQL Statement

Be aware there is an additional table called AE_STMT_B_TBL, which is a Statement Chunk Table. This is used to store the SQL statements entered in AE_STMT_TBL into smaller pieces or chunks once you save the definitions using the online panels. When an Application Engine program is executed, the chunks are selected and pieced together to form the original statement entered. This alleviates any incompatibility problems using Long datatypes in other databases. The synchronization between the AE_STMT_TBL and AE_STMT_B_TBL may become corrupted. An option does exist on the Application Definition panel which rebuilds the chunked statements. We'll identify this option in the pages ahead. Keep in mind that the breakdown of the SQL statements is done in the background.

As you begin constructing your Application Engine program through the online panels, the fields in each of these tables will be populated based on the selections you make. For example, a statement type of DO When sets the AE_DO_WHEN indicator in the AE_STEP_TBL. You have to fill in the name of your application, sections, and steps along with the descriptions. The statements themselves must also be filled in manually. Most of the remaining options are selected using radio buttons, drop-down lists, and checkboxes.

35.6 A/E DEFINITION PANELS

The Application Engine Definitions for each application, section, step, and statement are entered through a series of panels. You can navigate freely through these panels

using the folder tabs at the top or through some strategically placed push buttons. Let's take a look at the panels for each of the A/E categories (figures 35.1-35.4).

35.6.1 Application definition panel

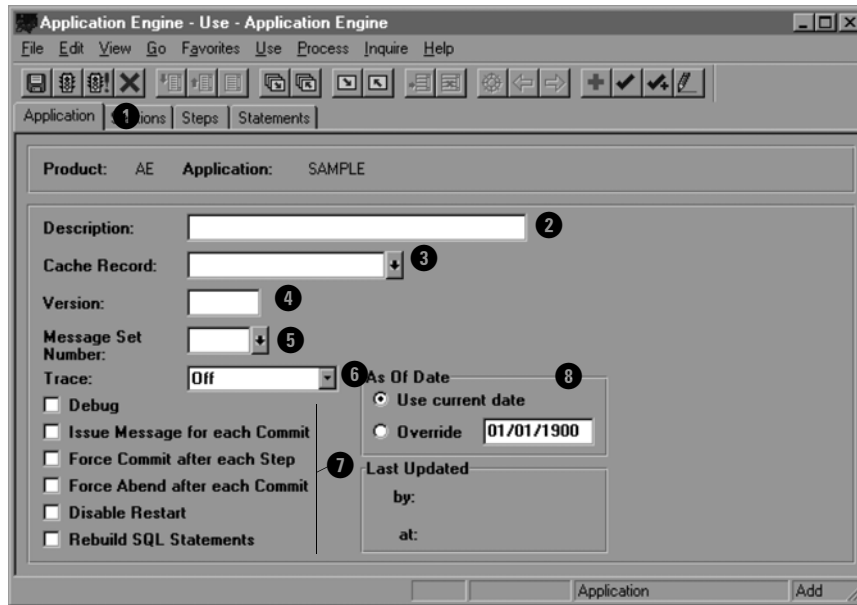


Figure 35.1 Application definition panel

- ❶ application folder tab, navigates through the four A/E panels
- ❷ description of application
- ❸ cache record used by application to store and pass values from one step to another
- ❹ version number (information only)
- ❺ default message set number, writes messages as our program progresses
- ❻ Trace options are used to create a trace file. Options are
 - Off* NO trace file produced
 - Steps Only* Each executed step is displayed on trace file showing time, section, step, and statement type.
 - SQL* In addition to Steps Only, the executed SQL is displayed on the trace file.
 - Abend Trap* Same information on trace file as SQL option. The trace file output is appended to any prior output for the same run rather than creating a new version. This creates a historical trace file that shows when an application prematurely aborted and was restarted.

- 7 Processing checkboxes are used to control the behavior of Application Engine.

Debug puts the application in interactive debugging mode. This allows you to set breakpoints, view the cache record contents, execute one step at a time, and issue commits and rollbacks.

Issue Message for each commit writes a message for each executed commit. It is recommended to use the trace option instead due to the volume of messages that may be produced by this feature.

Force Commit after each step instructs Application Engine to commit each step as the default method. Each section and step can override this if need be.

Force Abend after each commit is used for testing purposes. This is used to test your application restart capability. You can continually execute and restart the application until it is completed. If this cannot be done successfully, the program will need to be corrected.

Disable Restart allows you to restart your application from the beginning even if an abend occurs. Under normal circumstances a restart would be required. Use caution when using this option.

Rebuild SQL Statements is used to repopulate the Statement Chunk table (AE_STMT_B_TBL). When you create an SQL statement, it is stored in chunks. If you believe your chunked statements are out of sync with the statements entered, you can use this option to rebuild them.

- 8 The default date is used when filling in the effective date for each section. Sections are effective-dated. If development is spread out over several days, it's convenient to have the same effective date used when creating each new section.

35.6.2 Section definition panel

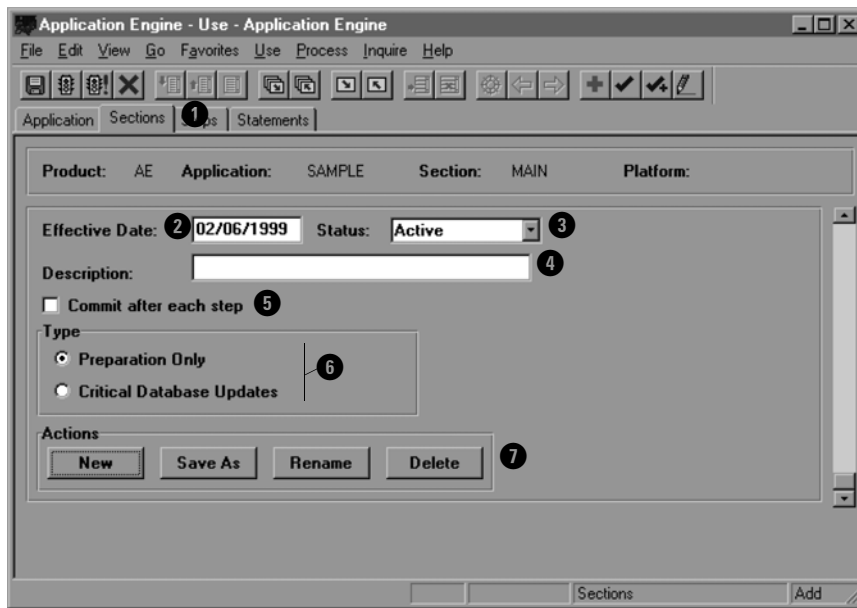


Figure 35.2 Section definition panel

- ❶ section folder tab, navigates through the four A/E panels
- ❷ effective date of the section
- ❸ effective status of the section
- ❹ description of application
- ❺ commit after each step within the section (This overrides the default setting.)
- ❻ Type option, used to designate the type of update being performed by the section

Critical Database Update should be used when a section could affect the integrity of the database in the event of an abend. A restart would be mandatory under these circumstances. Application Engine uses this indicator to update a column called AE_CRITICAL_PHASE in the AE_RUN_CONTROL table. This column will be set to 'Y' and can be used to determine if a restart is necessary.

Preparation Only simply means the section does not perform any critical database updates.

NOTE When trying to determine if a restart is necessary you can't rely totally on the AE_CRITICAL_PHASE indicator. If it is set to 'Y', you should definitely restart. If it does not equal 'Y', you may still need to restart your application. A prior step may have had critical database updates that need to be propagated in subsequent steps not yet executed. Extreme caution should be used to prevent integrity problems.

⑦ The action buttons manage your section development:

New adds a new section.

Save As can be used to copy the current section to a new name.

Rename renames the current section.

Delete deletes the current section.

35.6.3 Step definition panel

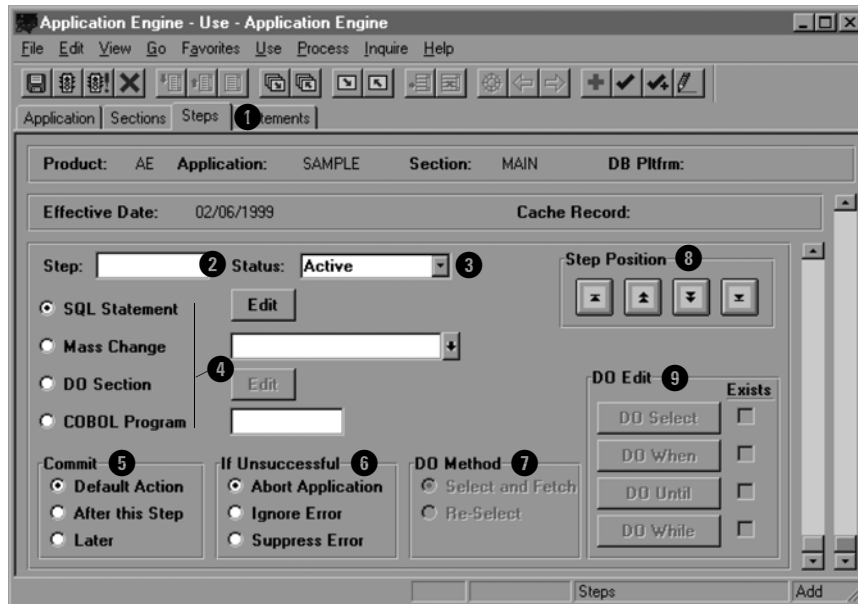


Figure 35.3 Step definition panel

- ① step folder tab, navigates through the four A/E panels
- ② the name of your step
- ③ effective status of the step

4 type of step

SQL Statement is used to perform an SQL or Application Engine statement entered in the Statements Panel. To access the Statements panel, press the SQL Statement Edit push button OR click on the Statement folder tab.

Mass Change allows you to execute a Mass Change program. The Mass Change definition ID can be selected from the drop-down list.

DO Section allows you to call another section. The called section can exist in your current application or an entirely different application. Sections can also be called dynamically at run time. Click on the DO Section Edit push button to access the DO Section properties dialog box. Here you will enter the DO section attributes.

COBOL Program allows a COBOL program to be called.

5 Commit override attributes for the current step

6 Error handling instructions

Abort Application performs a rollback and stops the process.

Ignore Error writes a message log entry and continues processing.

Suppress Error continues processing without any messages.

7 DO Method for DO Select statement types

The *Select and Fetch* method executes the DO Select statement once and fetches the rows one at a time. For each row fetched, the DO section is executed until all the rows have been processed.

The *Re-Select* method executes the DO Select statement and processes the first row fetched. After the first row is fetched, the DO section is executed. Upon returning, the DO Select statement is executed again, and if another row is returned, the DO section is executed again. This process continues until no rows are remaining.

8 step position buttons allow you to rearrange the order of steps in the section

9 alternate method to access DO Select statement types on the Statements panel

35.6.4 Statement definition panel

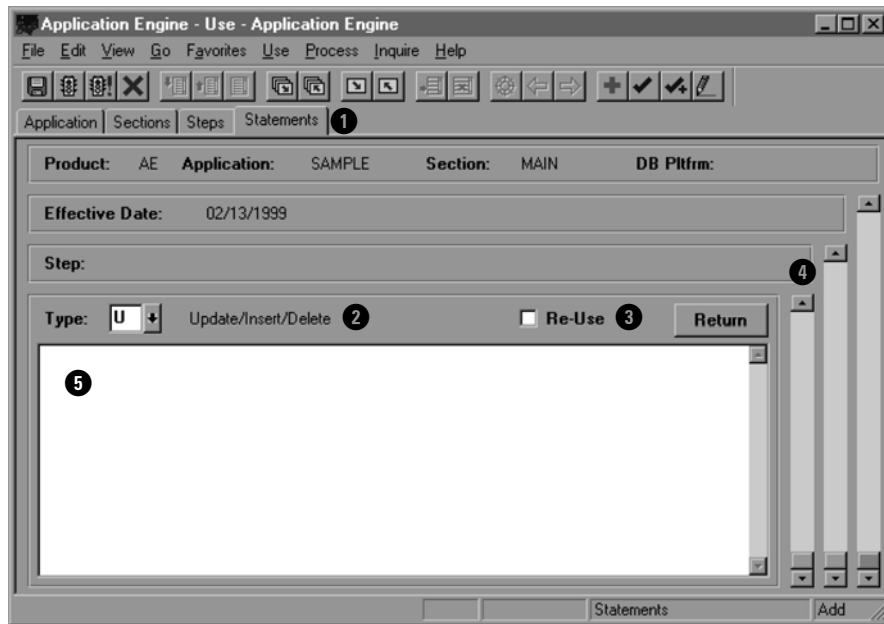


Figure 35.4 Statement definition panel

- ❶ Statement folder tab, navigates through the four A/E panels
- ❷ The type of statement to perform:

Comments is used to enter information about the step or to temporarily deactivate an executable statement.

Select is used to extract information and load it into your cache record.

Update statement types are used when executing SQL Updates, Inserts, or Deletes. It is also used when using the &MSG statement to write to the message log.

DO Select unconditionally executes a DO section for each row returned by the select SQL statement.

DO When conditionally executes a DO section. A *Select* statement is entered which will either return a row (representing a TRUE condition) or no rows (representing a FALSE condition). The DO section is executed when the condition is TRUE.

DO Until is used to break out of a DO Select. The DO section is performed and then the DO Until condition is evaluated. If a row is returned by the DO

Until Select statement, the DO section is no longer performed. If no rows are returned, the DO section is repeated.

DO While is similar to an SQR or COBOL *while* function. As long as the DO While Select statement returns a row (representing a TRUE condition), the DO section is performed. The DO While Select is executed again after the DO section has completed. This process continues until the DO While Select returns a FALSE condition (no rows returned).

- 3 To convert &BIND cache fields into true bind variables, use the Re-Use checkbox. A true bind variable means those designated by :1, :2, etc. You may have seen these in PeopleCode's `SQLExec` functions or stored SQL statements in COBOL.

NOTE	The PeopleSoft documentation does not provide a full explanation of this feature. It is used to improve performance by compiling the statement once and re-executing it with updated bind variable values. When using this feature, make sure the application is adequately tested with the desired results produced.
-------------	---

- 4 Click Return to go back to the Step Definition panel OR click on the Step folder tab.
- 5 Enter your SQL or Application Engine statements here.

No validation is performed on the SQL or Application Engine statement text entered. Any syntax errors will be identified at runtime. Proper testing is required for each step.

35.7 A/E SECTION/STEP RELATIONSHIP

All Application Engine programs begin with a section called MAIN. This can be considered the parent section when viewed in a hierarchical manner as depicted in figure 35.5. MAIN.STEP1 executes a section called LEVEL2A. This level has three steps. Once all three steps have been completed, control is passed back to MAIN, and MAIN.STEP2 is executed. This performs the LEVEL2B section. In turn, LEVEL2B.STEP1 performs LEVEL3. All called sections are performed in this manner until the last step in the MAIN section has been completed.

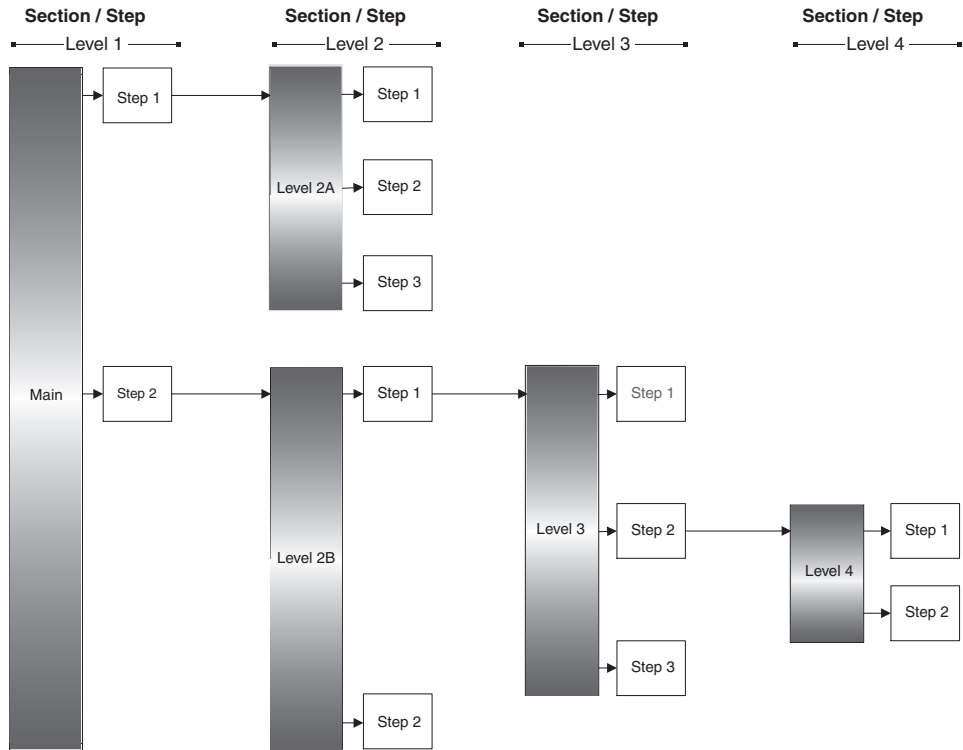


Figure 35.5 Section/step relationship

Let's take a look at the process flow (using the example in figure 35.5) as it would appear on a trace file listing. Each line represents a step executed within a section using the SECTION.STEP format:

```

MAIN . STEP1
  LEVEL2A . STEP1
  LEVEL2A . STEP2
  LEVEL2A . STEP3
MAIN . STEP2
  LEVEL2B . STEP1
    LEVEL3 . STEP1
    LEVEL3 . STEP2
      LEVEL4 . STEP1
      LEVEL4 . STEP2
    LEVEL3 . STEP3
  LEVEL2B . STEP2

```

This is the basic execution structure of an Application Engine program. Visualizing the process in this manner will help tremendously when creating a new program or modifying an existing one.

35.8 APPLICATION ENGINE: THE BIG PICTURE

If you look at the “big picture” in figure 35.6 you will see the heart of the Application Engine is the COBOL process PTPEMAIN. This is what controls each action being performed. When a process request is submitted, the PTPEMAIN program is called. It reads any parameters that may be assigned and automatically updates the cache record of your application. It reads and processes the A/E definitions you have created (application, sections, steps, statements). It compiles and executes SQL statements against the PeopleSoft tables specified. It inserts messages into the message log. The PTPEMAIN process also maintains a special Run Control record called AE_RUN_CONTROL which tracks the last committed step for restart purposes. PTPEMAIN handles all processing of Trace Files.

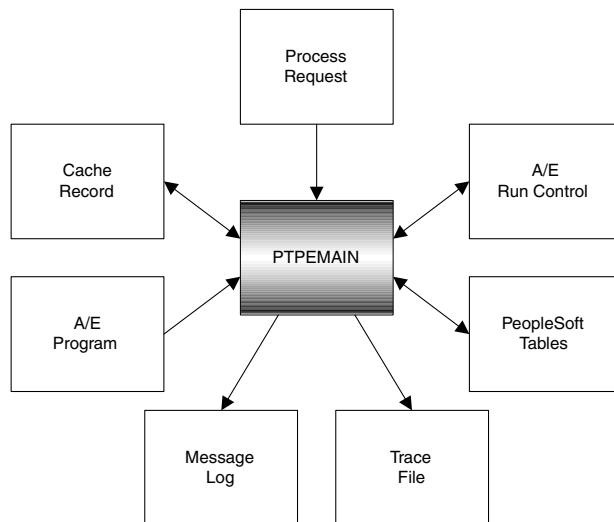


Figure 35.6 PTPEMAIN is the “heart” of Application Engine



CHAPTER 36

Build your first application

36.1 Before we begin: an introduction to our tutorial	789	36.4 Beginning our tutorial	797
36.2 Adding message catalog entries	790	36.5 Exercise 1: Hello World!	797
36.3 Creating a custom cache record	793	36.6 SQR/Application Engine comparison	805

36.1 BEFORE WE BEGIN: AN INTRODUCTION TO OUR TUTORIAL

As discussed earlier, two of the elements of an Application Engine program are the message catalog and the cache record. It's not necessary to build these from scratch. Any predefined cache record within PeopleSoft can be used as can any existing message set. You can also modify the existing cache record or message set to support your particular requirements. For our purposes, we'll create a new cache record and a new message set.

36.2 ADDING MESSAGE CATALOG ENTRIES

Before we begin developing any custom applications, let's create our own custom message set in the Message Catalog table. The message set number will be linked to our Application Engine programs.

Navigation: Go →PeopleTools →Utilities →Message Catalog →Add

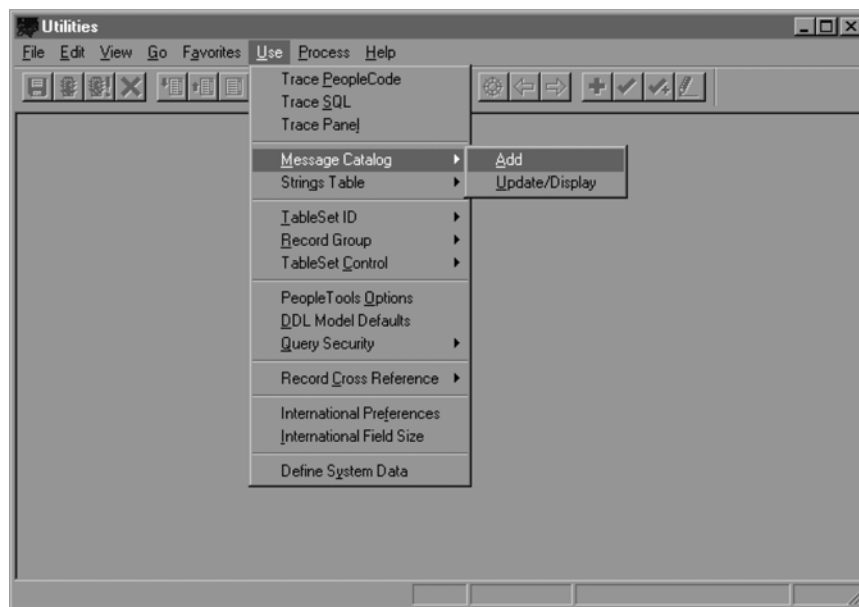


Figure 36.1 Adding a message catalog entry

Note that PeopleSoft reserves message set numbers up to 20000. When adding a custom message set, utilize any available number after 20000. This will make upgrades go much smoother. For our custom message set number, we'll be using 20001. English will be the language code for our custom applications (figure 36.2).



Figure 36.2 Adding Message Set Number 20001

We'll use the description "User Messages" for the 20001 message set. For each message we add to the message set, we'll assign a sequential number starting from 1

(figure 36.3). This is the message number we'll use to specify which message text and parameter format to use.

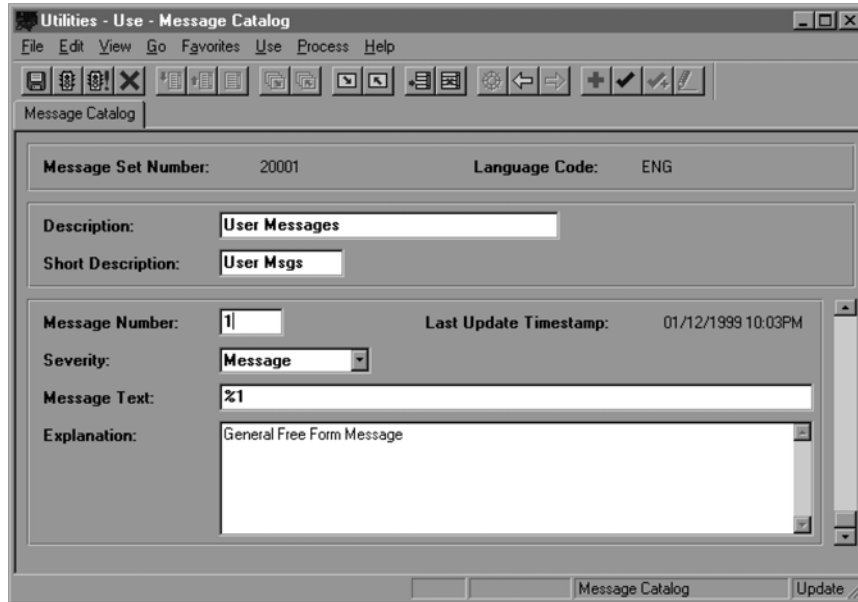


Figure 36.3 Message 1 definition

The %1 in the body of the message text indicates a value will be passed as a parameter and substituted in its place. We will use this in our first application we create. We are also adding two more messages to the message set that we'll be using in subsequent exercises. To add more messages, place the cursor in the message number edit box and press the F7 key to insert a new row. Use the scroll bar to view the messages in the message set.

Message Number 2 of our message set contains two input parameters %1 and %2 (figure 36.4).

Message Number 3 has two input parameters as well as actual text in the body of the message text area (figure 36.5). As you may have guessed by the Text and Explanation, we will be writing an application to select a table and display the number of rows.

Utilities - Use - Message Catalog

File Edit View Go Favorites Use Process Help

Message Catalog

Message Set Number: 20001 Language Code: ENG

Description: User Messages

Short Description: User Msgs

Message Number: 2 Last Update Timestamp: 01/12/1999 10:05PM

Severity: Message

Message Text: %1 %2

Explanation: Free Form - 2 parameter message

Message Catalog Update

Figure 36.4 Message 2 definition

Utilities - Use - Message Catalog

File Edit View Go Favorites Use Process Help

Message Catalog

Message Set Number: 20001 Language Code: ENG

Description: User Messages

Short Description: User Msgs

Message Number: 3 Last Update Timestamp: 01/12/1999 10:08PM

Severity: Message

Message Text: %1 contains %2 records

Explanation: Table / Count Message

Message Catalog Update

Figure 36.5 Message 3 definition

36.3 CREATING A CUSTOM CACHE RECORD

A cache record is no different than any other PeopleSoft record (or work record) you create in Application Designer. A couple of simple rules must be followed: there can only be one key field, and that key field has to be `PROCESS_INSTANCE`. When your Application Engine program is executed, a process instance is assigned by the Process Scheduler. This process instance is used to store the cache fields for your job, as opposed to someone else's (which has its own process instance).

Navigation: Go →PeopleTools →Application Designer →File →New...

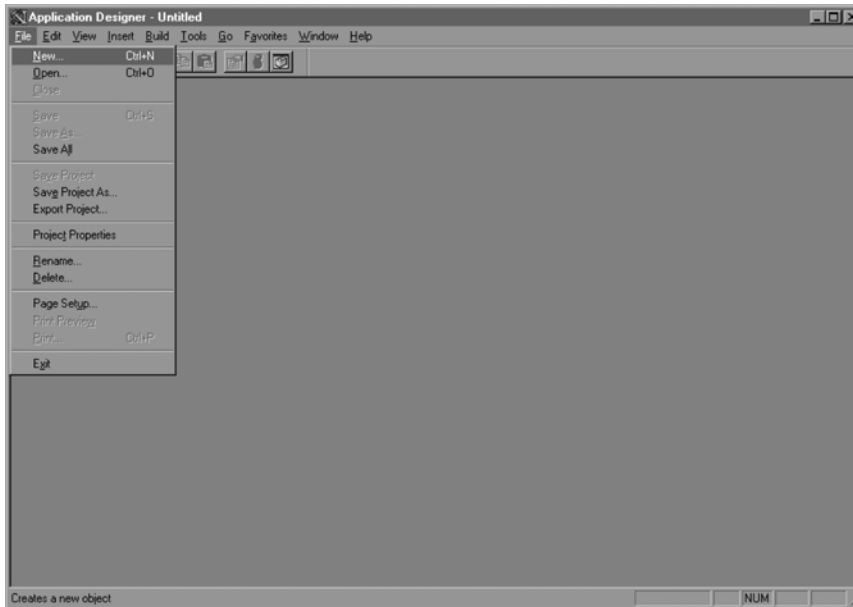


Figure 36.6 Creating a new object using Application Designer



Figure 36.7
Creating a new record object

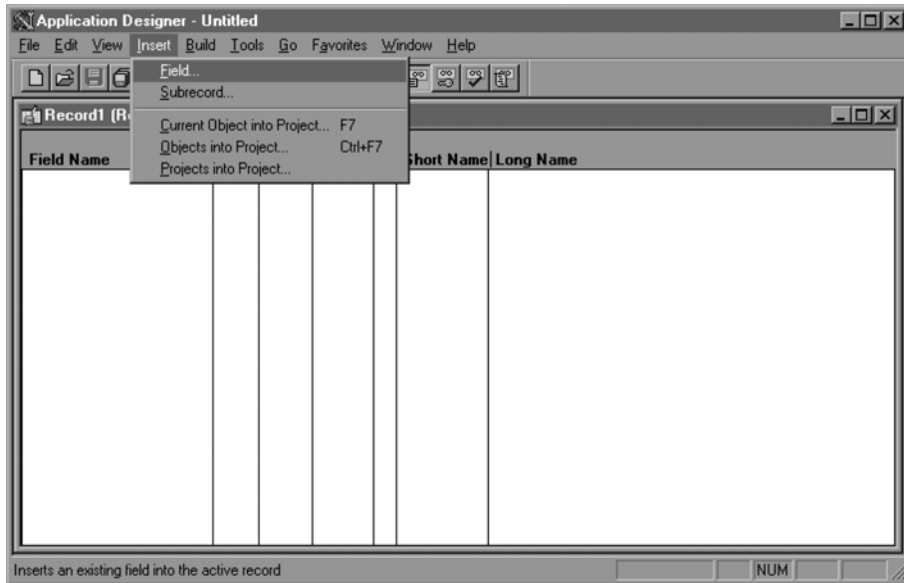


Figure 36.8 Inserting existing fields into the new record

Now we're going to add the following fields to our custom cache record: PROCESS_INSTANCE (Process Instance), COUNTER (Generic Counter), RECNAME (Record (Table) Name), FIELDNAME (Field Name), and AE_DECIDE (Decision Switch).

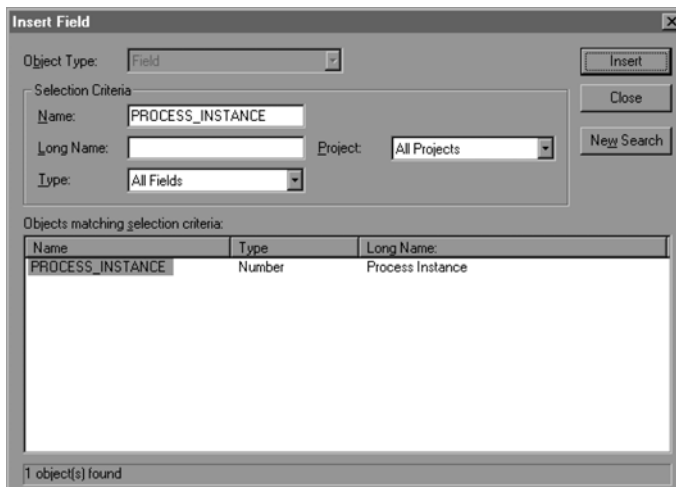


Figure 36.9 Inserting the process instance field object

The purpose of these fields will become evident as we build our applications.

NOTE Only one key field can exist on a cache record—PROCESS_INSTANCE!

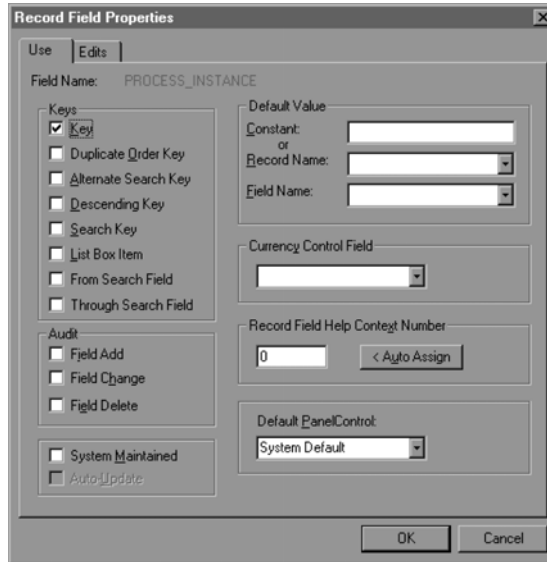


Figure 36.10
Assigning the primary key

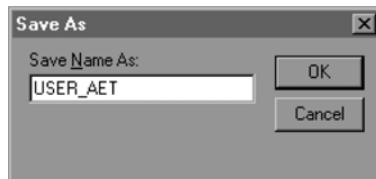


Figure 36.11 Saving the record

Once the fields have been added and the primary key (PROCESS_INSTANCE) assigned (figure 36.10), we can save the record (figure 36.11).

We'll call our custom cache record USER_AET.

Now, we need to create the table within the database itself using SQL*Create. This isn't necessary if the cache record is a work record. For the purposes of this book, we will use a physical SQL table for the cache record. When a work record is used, the cache values are lost if the program ends or aborts. You will not be able to restart an aborted process using a work record, therefore, it is a good practice to avoid them and use a physical SQL table.

While the USER_AET record is still open, click on the Build menu item, then click on Current Object (figure 36.12).

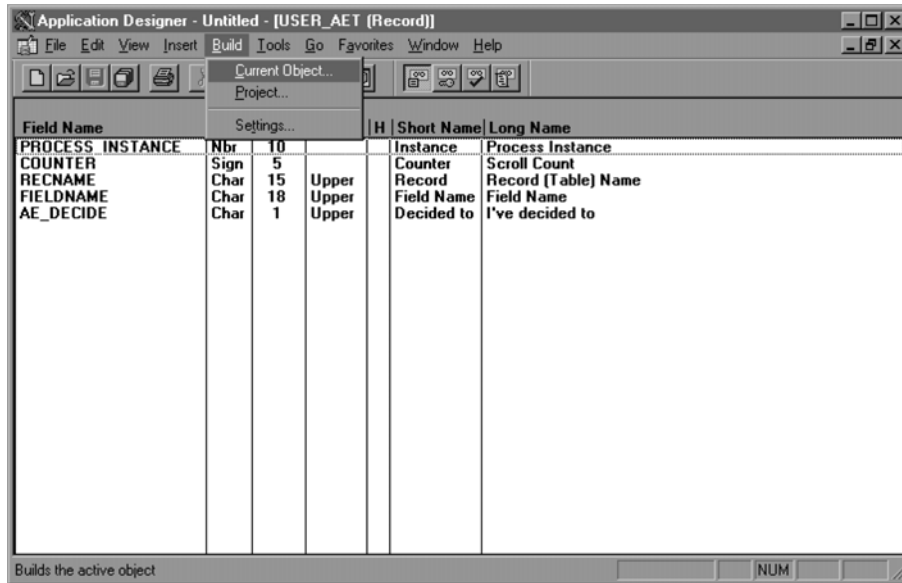


Figure 36.12 Building the current object in the database

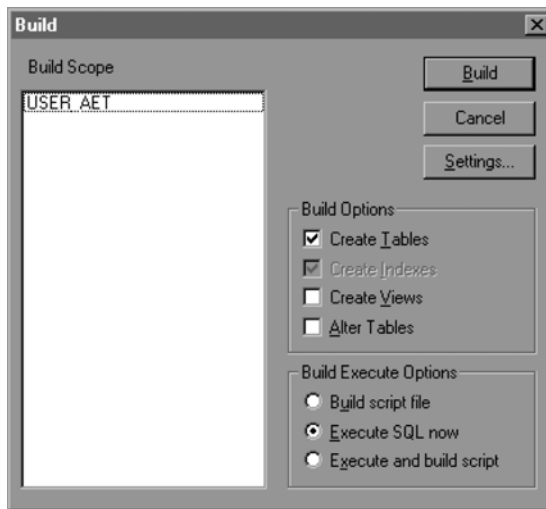


Figure 36.13
Executing SQL table creation

The Build screen appears with our current object, USER_AET, in the selection box. Click on the Create Tables checkbox and the Execute SQL now radio button. Now click on the Build push button to create the USER_AET table at the database level.

Our cache record is complete. We're ready for our first Application Engine program.

36.4 BEGINNING OUR TUTORIAL

Our first application isn't original but will clearly demonstrate basic Application Engine functionality. In many books about programming languages, it's customary to begin with an exercise that displays the short phrase "Hello world." This book is no exception. Let's begin.

36.5 EXERCISE 1: HELLO WORLD!

36.5.1 Creating an SQR version

Let's start by writing a simple SQR that displays "Hello World" on the log file:

```
! USER001.SQR
begin-program
do Main-Step1
end-program
begin-procedure Main-Step1
show 'Hello World'
end-procedure
```

After execution the SQR.log looks like this:

```
Hello World
```

Now, let's create a version of the program using Application Engine.

36.5.2 Defining the application

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine
→Application →Add

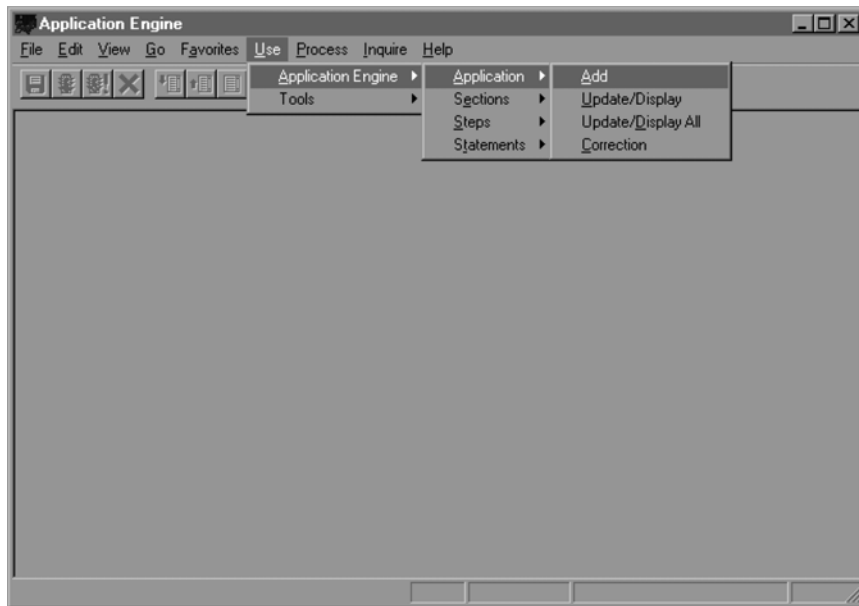


Figure 36.14 Creating a new application

Our first step here is to add a fully qualified program name. Application Engine programs are identified by *product* and *application*. Some examples of products are HR, Accounts Receivable, and General Ledger. Product categories are used to logically group your Application Engine programs.

Let's select the Application Engine Product type (PS/AE). We'll call our first application USER001 (figure 36.15). Remember, all Application Engine programs must begin with a section called MAIN. Additional sections may be added if necessary. Since we don't plan on using any database specific functionality, we can leave database platform blank.

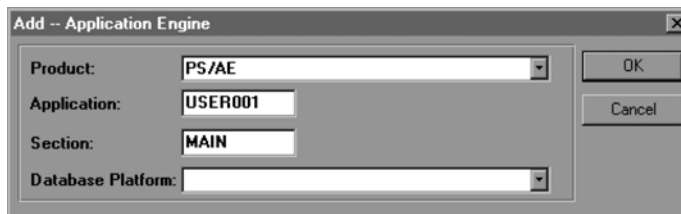


Figure 36.15
Naming the application

The screen in figure 36.16 shows the Application Definition screen for our new program USER001. Notice the cache record and message set number edit boxes. We'll use our custom cache record USER_AET and custom message set 20,001. We can also document the program version using the Version edit box.



Figure 36.16 Defining our application

Take note of the group box As Of Date. It is set to use the current date when adding any new sections. When program development is spread out over several days, it's a good idea to override the current date with a common date for all sections.

We can now move to the Section MAIN definition.

TIP Application Engine requires a section called MAIN in every program. The MAIN section is always executed first.

36.5.3 Creating sections, steps, and statements

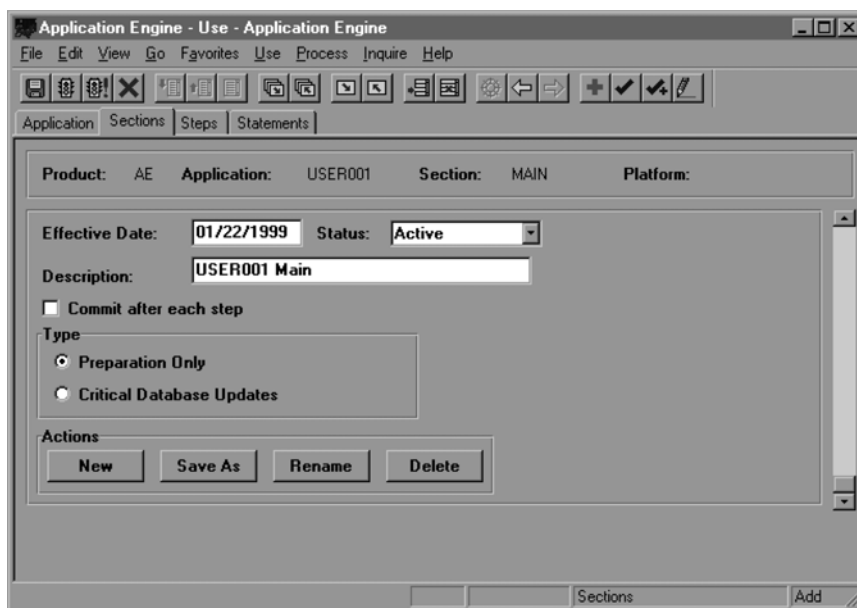


Figure 36.17 Defining section MAIN

The only additional piece of information we need to include on the section MAIN definition is the description. We're ready to add a step to our application.

Our first and only step of the application will be called STEP1 (figure 36.18). Since the objective of this step is to write a message, we use the SQL statement type. The message function of Application Engine comes under the SQL statement category. Notice the additional step options. In some cases, a Mass Change program, another A/E Section, or a COBOL program can be used instead of an SQL statement.

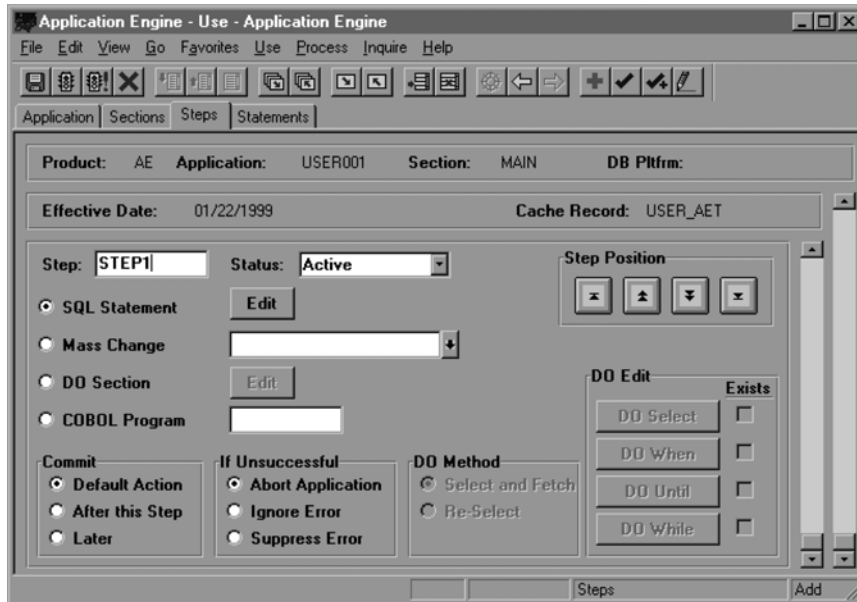


Figure 36.18 Defining our first step

36.5.4 Introducing the &MSG function

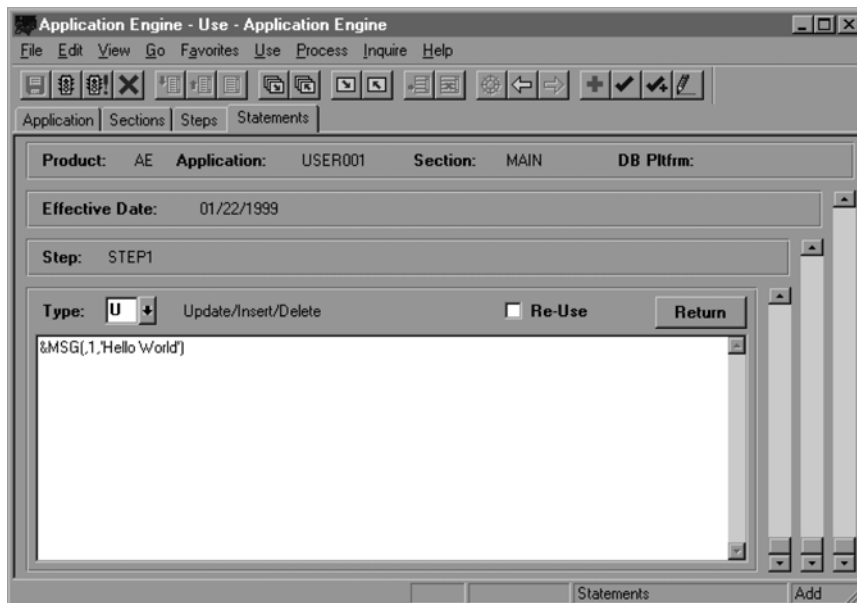


Figure 36.19 Defining our first statement

The &MSG function writes a message to the Message Log using the following format:

```
&MSG( [Message_Set_Number], Message_Number, [Parm_1],... [Parm_n] )
```

The &MSG function always uses an SQL statement type of UPDATE and must be the first and only function or command in the statement.

For more information on the &MSG function refer to the function reference in appendix F.

In our &MSG function, we do not specify a message set, thereby defaulting to the 20001 message set we've defined on the application definition. The message number is 1, which you may recall consisted of a lone %1 input parameter. The "Hello World" text is the input parameter we are passing. We have completed our first Application Engine program! If all goes as planned, the "Hello World" phrase will appear in the message log.

We're ready to test the USER001 Application.

36.5.5 Running an Application Engine program

Navigation: Go →PeopleTools →Application Engine →Process →Request →Request →Add

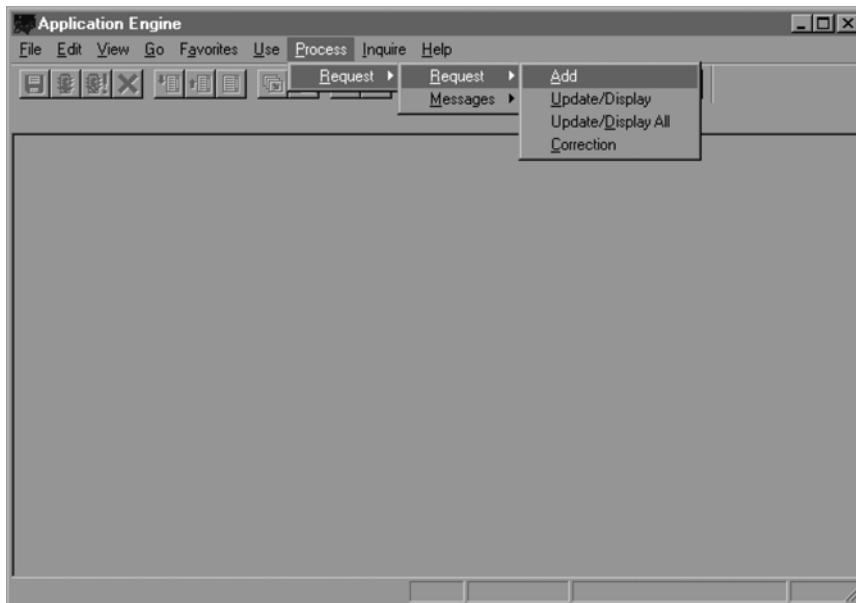


Figure 36.20 Adding a Process Request

We can test our Application Engine program through the Process Request panel. This provides a means to test without having to set up individual Run Control panels for each program we create. We also don't have to create any Process Scheduler definitions. Simply enter a Run Control ID and execute your program!

In figure 36.21, we assign a Run Control ID of #USER001.



Figure 36.21
Assigning a Run Control ID

On the Process Request panel, we enter the product and application of our program (figure 36.22). The bottom half of the screen is used to initialize fields on the cache record. We'll discuss these in a future exercise. Our first application doesn't utilize any cache fields directly.

Once the Process Request panel is populated correctly, click on Traffic Signal to initiate a Process Scheduler request.

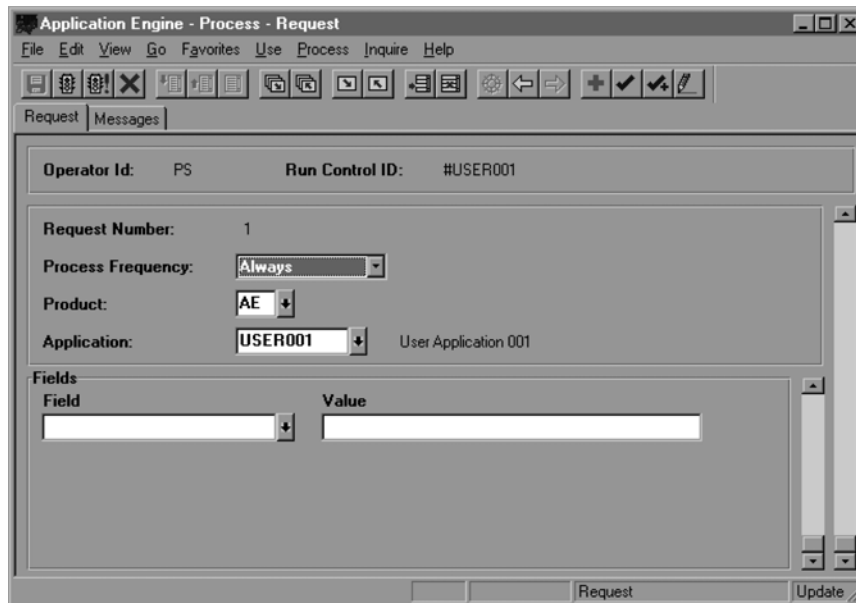


Figure 36.22 Defining a Process request

Highlight the Application Engine AEADHOC Process, and click the OK button to start (figure 36.23).

Description	Name	Process Type Descr
Application Engine	AEADHOC	Application Engine
Application Engine	PTPEMAIN	COBOL SQL

Figure 36.23 Submitting a Process Scheduler request

NOTE You may find two process definitions in the Process Scheduler Request panel. Both AEADHOC and PTPEMAIN are linked to the Process Request panel.

Because AEADHOC is defined as an Application Engine process type, it will call the PTPEMAIN program by default. We could choose either to test our application. For purposes of this book, we'll use the same process definition throughout our exercises. We'll choose AEADHOC since this demonstrates a link to PTPEMAIN, which we'll also create in our last exercise.

Although you can run Application Engine processes on both the client and server, we will run on the client throughout this book.

36.5.6 Reviewing Application Engine messages

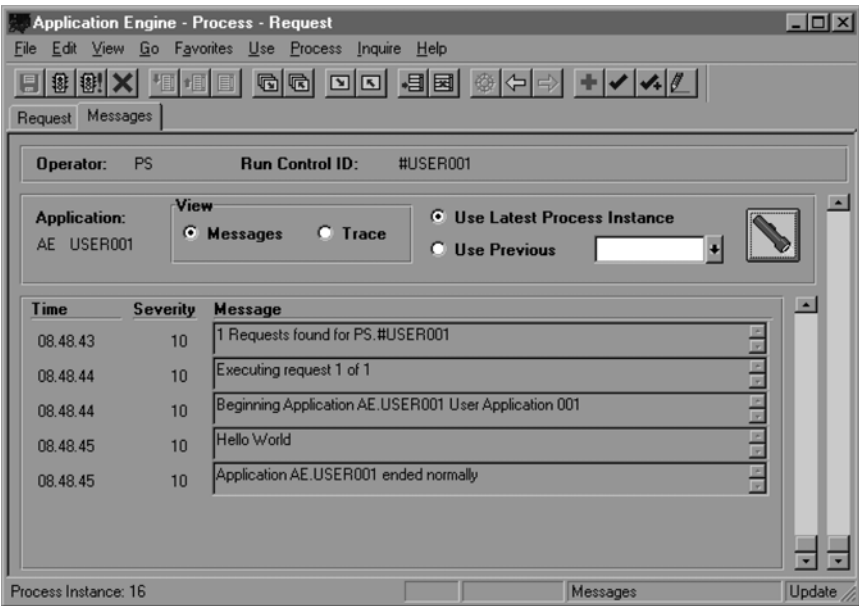


Figure 36.24 Reviewing Process Request messages

To view the message log, click on the Message folder tab. Set the View Messages radio button and the Use Latest Process Instance radio button. Click on the Flashlight to display the Latest Process Instance Message Log which happens to be our first run. At precisely 08.48.45, the “Hello World” message was displayed (figure 36.24). Our first program was successful!

36.6 SQR/APPLICATION ENGINE COMPARISON

Let’s take a look at the logical structure of both our programs.

SQR:

Begin-Program
Main-Step1

Application Engine:

USER001
MAIN.STEP1

Both programs follow the same structure: a step which writes a message, is performed. This comparison should prove to be beneficial as our exercises become more complex.

KEY POINTS

- 1 Two important elements of an Application Engine program are the message catalog and the cache record.
- 2 A cache record can have only one key, the process instance. The cache record will be used to store and retrieve values similar to using variables in conventional programs.
- 3 A cache record is assigned to your A/E program. The cache record is used to store and pass values from one step to another.
- 4 All Application Engine programs begin with a section called MAIN. Sections are similar to procedures in COBOL or SQL.
- 5 Each unit of work is broken down into a step. A step can call other sections, a COBOL program, a mass change program, or an SQL statement within the step itself.
- 6 Statements can be native SQL statements or Application Engine functions (or in some cases a combination of the two).
- 7 The &MSG function allows you to monitor the progress of your program by writing messages to the message log.
- 8 You can test your Application Engine programs using the Process Request panel.
- 9 The Process Request Messages panel allows you to view the messages generated during the run. Some messages are due to the &MSG function in your program while others are written by the PTPEMAIN process automatically.



C H A P T E R 37

Using cache fields

- 37.1 Exercise 2: How many rows in PERSONAL_DATA? 807
37.2 SQR/Application Engine comparison 816

Variables do not exist in an Application Engine. Values are stored in a cache field and utilized by subsequent steps. Our next exercise demonstrates the use of cache fields.

37.1 EXERCISE 2: HOW MANY ROWS IN PERSONAL_DATA?

Our next exercise is also a basic one. We're going to select the number of rows in the PERSONAL_DATA table and store the count in the cache record. The next step will utilize the cached value and display the results in the message log.

PERSONAL_DATA is a core table in the PeopleSoft HRMS application. You may substitute any table for this exercise. For example, if you're running PeopleSoft Accounts Receivable you may want to use the CUSTOMER table instead.

37.1.1 Creating an SQR version

We'll begin this exercise by displaying the SQR version of this program:

```
! USER002.SQR

begin-program

do Main-Step1
do Main-Step2

end-program

begin-procedure Main-Step1

let #counter = 0

begin-select

count(*)          &counter

  let #counter = &counter

  from ps_personal_data

end-select

end-procedure

begin-procedure Main-Step2

show 'PERSONAL_DATA Record Count: ' #counter

end-procedure
```

The first procedure, Main-Step1, populates the variable #counter with the number of rows in the PERSONAL_DATA table. The second procedure, Main-Step2, displays the results.

After execution, the SQR.log looks like this:

```
PERSONAL_DATA Record Count: 347.000000
```

Now, let's create a version of the program using Application Engine.

Our second Application Engine program (figure 37.1) is called USER002, and, as we've learned, must begin with the section MAIN.

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine
→Application →Add

Figure 37.1
Naming the application

Once again, we use the USER_AET cache record and the 20001 message set number (figure 37.2).

Figure 37.2 Defining our application

Figure 37.3 shows the section definition for MAIN.

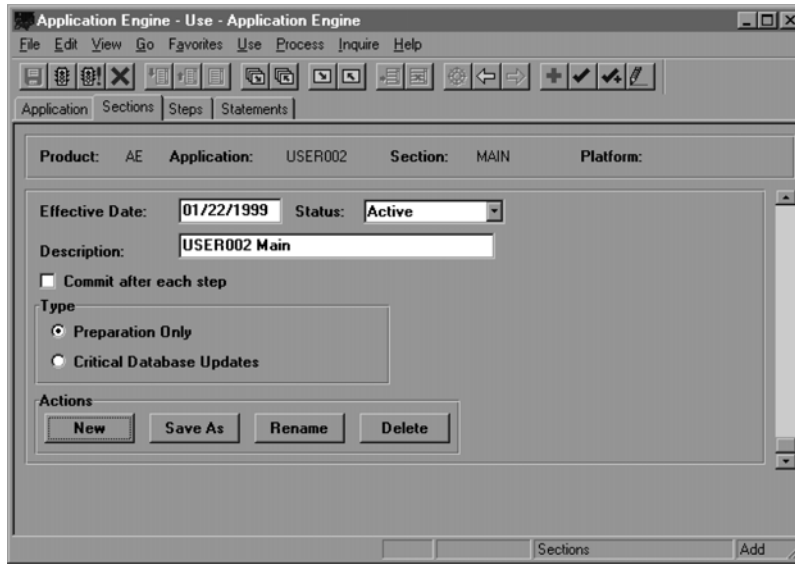


Figure 37.3 Defining section MAIN

Our first step is called STEP1 (figure 37.4). It consists of one SQL statement, which selects the number of rows in the PERSONAL_DATA table and stores the result in our cache field COUNTER.

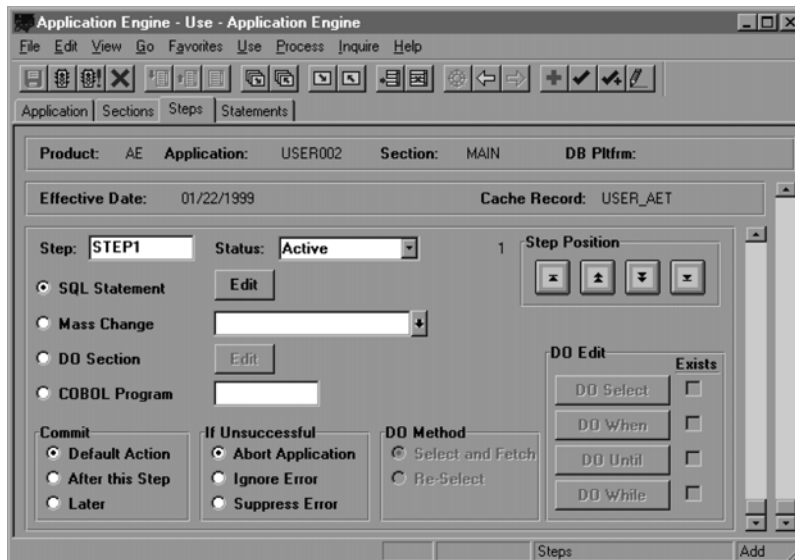


Figure 37.4 Defining STEP1

37.1.2 Assigning cache fields values with &SELECT

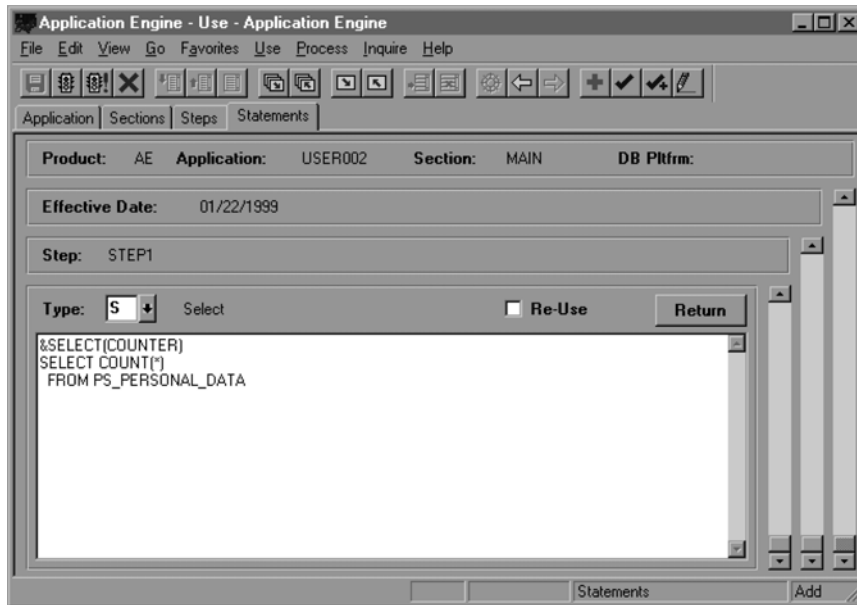


Figure 37.5 Entering the SQL statement text

We begin our statement definition by assigning a statement type of `SELECT`. Our statement text looks like this:

```
&SELECT(COUNTER)

SELECT COUNT(*)

FROM PS_PERSONAL_DATA
```

The first line uses the Application Engine function `&SELECT`. This function updates the cache field with the value assigned by the corresponding SQL `SELECT` statement. The `&SELECT` function has the following format:

```
&SELECT(cache_field_1 [,cache_field_2] [,cache_field_x] )
SELECT field_1 [,field_2] [,field_x] )
```

`&SELECT` is immediately followed by an SQL `Select` statement.

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.

For more information on the &SELECT statement, refer to the function reference in appendix F.

37.1.3 Defining multiple steps within a section

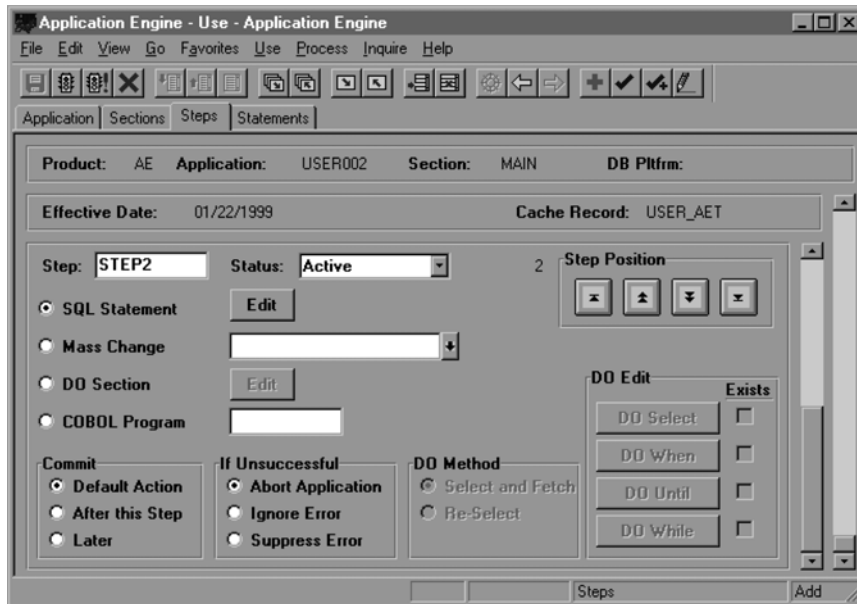


Figure 37.6 Defining STEP2

Our second step, STEP2, accesses the cache field COUNTER and displays the results in the message log. To create the new step, place the cursor in the step edit box and press the F7 key to insert a new row. When the step panel is complete, press the statement folder tab to enter our statement.

37.1.4 Retrieving cache field values with &BIND

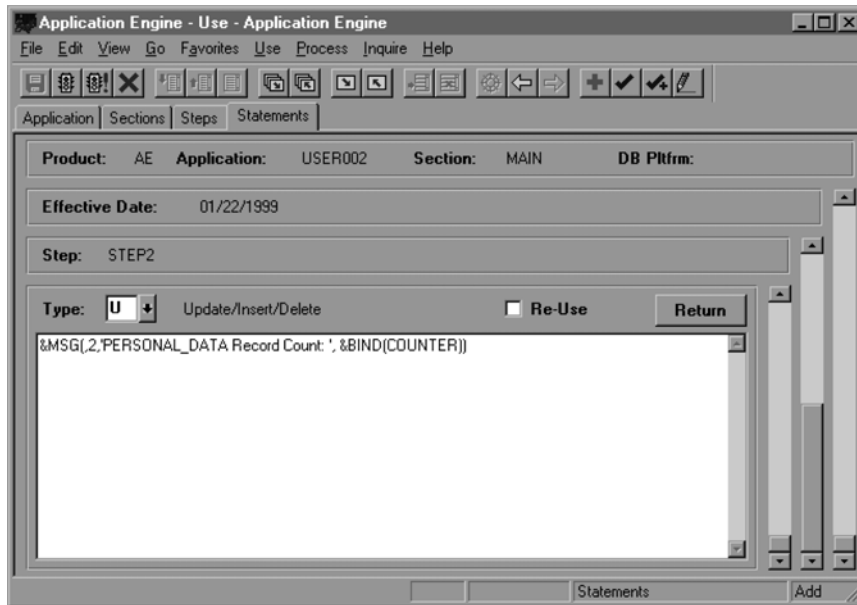


Figure 37.7 Entering &MSG statement text

As we did in exercise 1, we utilize the &MSG Application Engine function to write to the message log. We're not specifying a message set, so we default to the 20001 message set we've defined on the Application Definition panel. Message 2 was defined using two input parameters, %1 and %2. The first parameter we're passing is the string 'PERSONAL_DATA Record Count: '. The second parameter is the value stored in our cache field COUNTER. To retrieve the assigned cache field value, we use another Application Engine function called &BIND. This function has the following format:

```
&BIND(cache_field [,NOQUOTES] [,NOWRAP] [,STATIC])
```

The &BIND function follows these 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 quotation marks unless the optional NOQUOTES parameter is used.
- Date fields will be 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 will resolve the `&BIND` variable before compiling the SQL statement. This is useful when creating Dynamic SQL statements.

We'll discuss Dynamic SQL in the next exercise. Also, you can refer to the function reference in appendix F for more information on the `&BIND` function.

We're now ready to test `USER002`.

We test our new application using the Process Request Panel. We'll use `#USER002` as our Run Control ID (figure 37.8).

Navigation: Go → PeopleTools → Application Engine → Process → Request → Request → Add



Figure 37.8
Assigning a Run Control ID

Simply fill in the `PRODUCT/APPLICATION` without any cache field values. Click on Traffic Signal to initiate a Process Scheduler request.

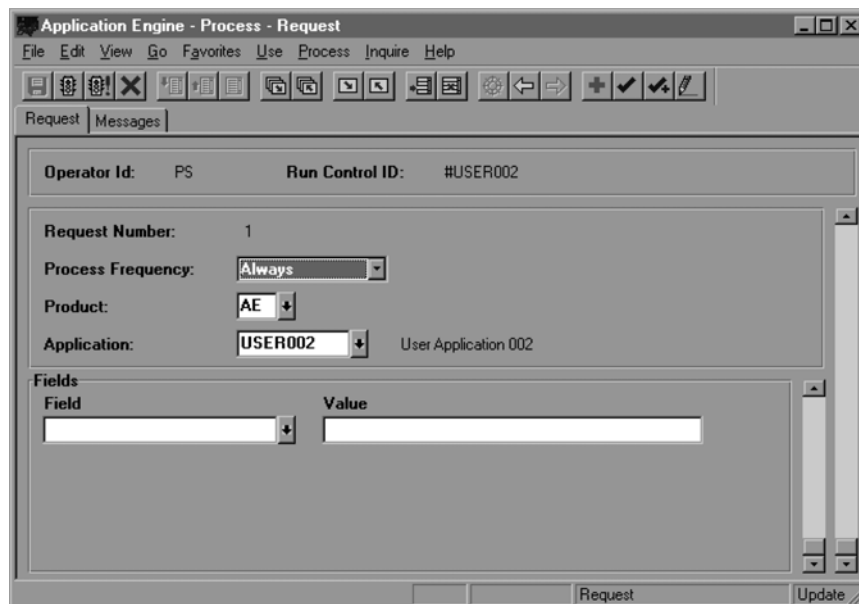
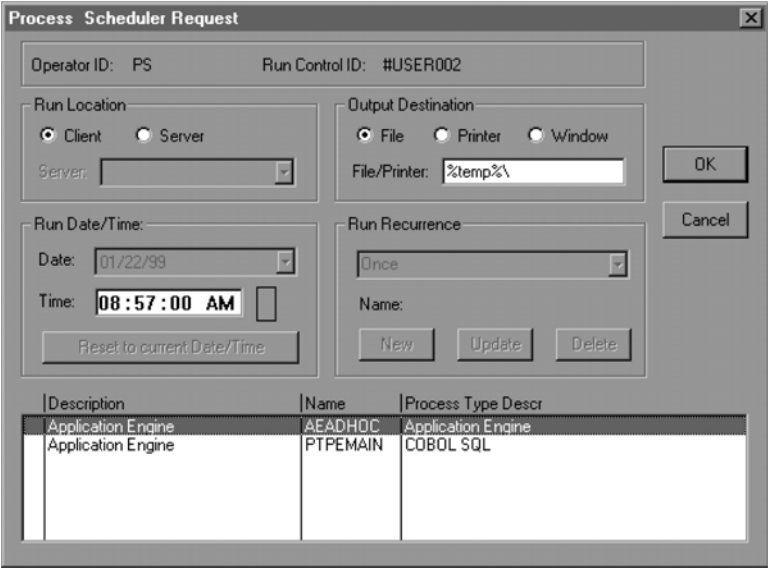


Figure 37.9 Defining a process request

Highlight the Application Engine AEADHOC process and click the OK button to start (figure 37.10).



The 'Process Scheduler Request' dialog box contains the following fields and controls:

- Operator ID:** PS
- Run Control ID:** #USER002
- Run Location:** ☒ Client ☐ Server. A 'Server:' dropdown is visible below the radio buttons.
- Output Destination:** ☒ File ☐ Printer ☐ Window. A text field 'File/Printer:' contains '%temp%\'. Buttons 'OK' and 'Cancel' are to the right.
- Run Date/Time:** Date: 01/22/99, Time: 08:57:00 AM. A 'Reset to current Date/Time' button is below.
- Run Recurrence:** A dropdown menu set to 'Once'. Below it are 'New', 'Update', and 'Delete' buttons.
- Table:** A table with 3 columns: Description, Name, and Process Type Descr.

Description	Name	Process Type Descr
Application Engine	AEADHOC	Application Engine
Application Engine	PTPEMAIN	COBOL SQL

Figure 37.10 Submitting a Process Scheduler request



The 'Application Engine - Process - Request' window displays the following information:

- Operator:** PS
- Run Control ID:** #USER002
- Application:** AE USER002
- View:** ☒ Messages ☐ Trace
- Options:** ☒ Use Latest Process Instance, ☐ Use Previous (with a dropdown menu).
- Message Log Table:**

Time	Severity	Message
08:58:12	10	1 Requests found for PS.#USER002
08:58:12	10	Executing request 1 of 1
08:58:13	10	Beginning Application AE.USER002 User Application 002
08:58:15	10	PERSONAL_DATA Record Count: 347
08:58:16	10	Application AE.USER002 ended normally
- Status Bar:** Process Instance: 17, Messages, Update button.

Figure 37.11 Reviewing process request messages

To view the message log, click on the Message Folder tab. Set the View Messages radio button and the Use Latest Process Instance radio button. Click on the Flashlight to display the Latest Process Instance Message Log for the current run. Our message appears in the log with the same record count as our SQR version of the program. Another success!

37.2 ***SQR/APPLICATION ENGINE COMPARISON***

Let's take a look at the logical structure of both our programs:

SQR:

```
Begin-Program
Main-Step1
Main-Step2
```

Application Engine:

```
USER002
MAIN.STEP1
MAIN.STEP2
```

Once again, the structures are the same. Step 1 in both programs retrieves the number of rows in PERSONAL_DATA and displays the results in step 2.

KEY POINTS

Exercise 2 demonstrated some key features in Application Engine:

- 1 Multiple Steps may be defined within a section.
- 2 The &SELECT function is used in tandem with an SQL `Select` to assign values to cache record fields.
- 3 Cache record field values are retrieved with the &BIND function. The &BIND function can be used within SQL statements as bind variables or to create dynamic SQL.



C H A P T E R 3 8

Dynamic SQL statements

38.1 Exercise 3: How many rows in any table? 817

38.2 SQR/Application Engine comparison 826

38.1 EXERCISE 3: HOW MANY ROWS IN ANY TABLE?

In our previous exercise, we determined the number of rows in PERSONAL_DATA.

In this exercise, we display the number of rows in ANY table. When we created our USER_AET cache record, we included the field RECNAME. This cache field is populated on the Process Request Panel with the name of the record we want to utilize. Using the RECNAME value, we select and store the resulting count in the cache record. The next step will utilize the cached value and display the results in the message log as we did in the prior exercise.

38.1.1 Creating an SQR version

We begin this exercise by displaying the SQR version of this program:

Listing 38.1

USER003.sqr

```
! USER003.SQR

begin-program

input $recname 'Enter RECNAME' maxlen=15

do Main-Step1
do Main-Step2

end-program

begin-procedure Main-Step1

let $table  = 'ps_' || $recname
let #counter = 0

begin-select

count(*)      &counter

    let #counter = &counter

    from [$table]

end-select

end-procedure

begin-procedure Main-Step2

show ' '

show $recname ' Record Count: ' #counter

end-procedure
```

The user is prompted for a record name using the INPUT statement. Note there is no validation on the entered value. We are assuming valid input to keep the program simple. The first procedure Main-Step1 populates the variable #counter with the number of rows in the table specified by the user (SQR also supports Dynamic SQL). The second procedure Main-Step2 displays the results. For our example, we use the JOB table as our RECNAME value.

After execution the SQR.log looks like this:

```
Enter RECNAME: JOB

JOB Record Count: 1685.000000
```


Now, let's create a version of the program using Application Engine.

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine →Application →Add

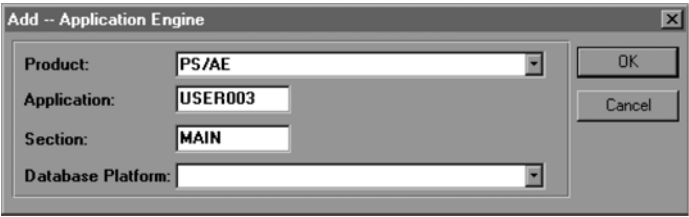


Figure 38.1
Naming the application

Our third Application Engine program is called ‘USER003’ and starts with section MAIN.

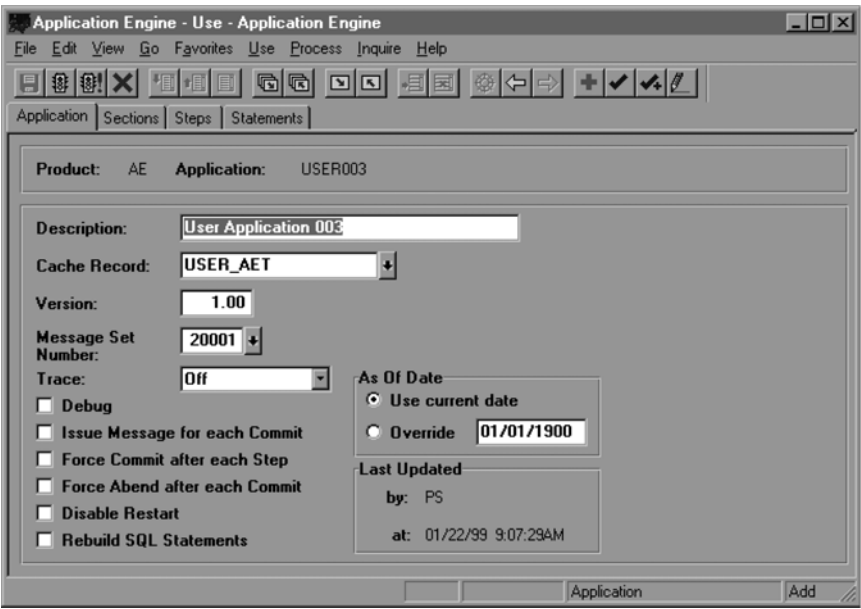


Figure 38.2 Defining the application

We'll continue to use the USER_AET cache record as well as the 20,001 message set. We'll fill in the description of our section (figure 38.3) and proceed with the first step.

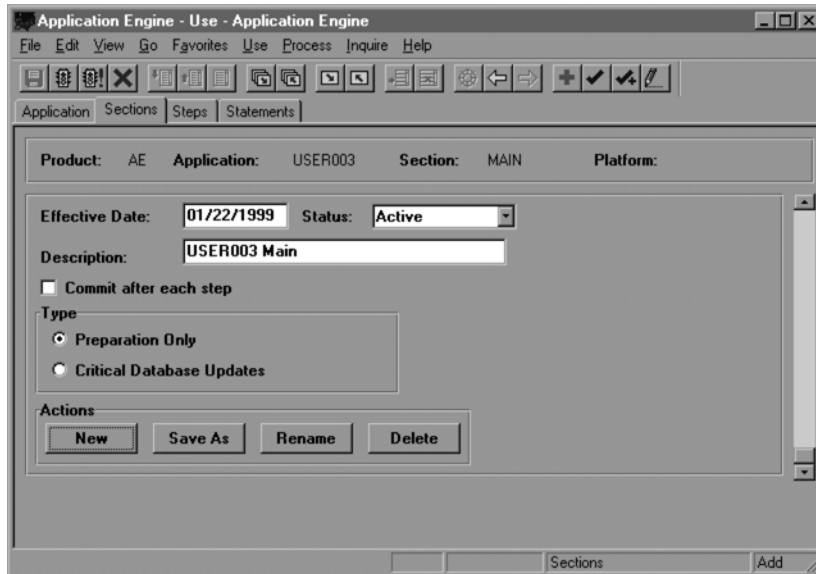


Figure 38.3 Defining section MAIN

We'll call our first step STEP1 (figure 38.4) and move to the Statement Definition panel.

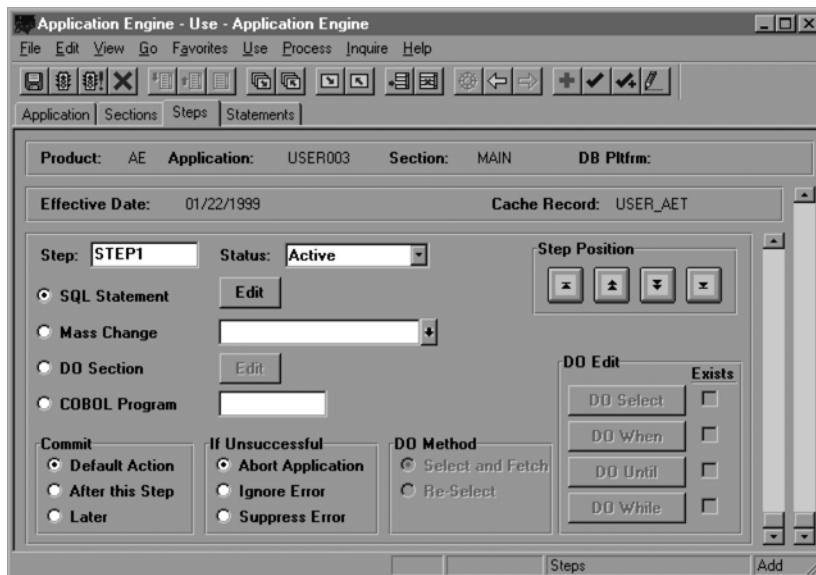


Figure 38.4 Defining STEP1

38.1.2 Using &BIND parameters NOQUOTES and STATIC

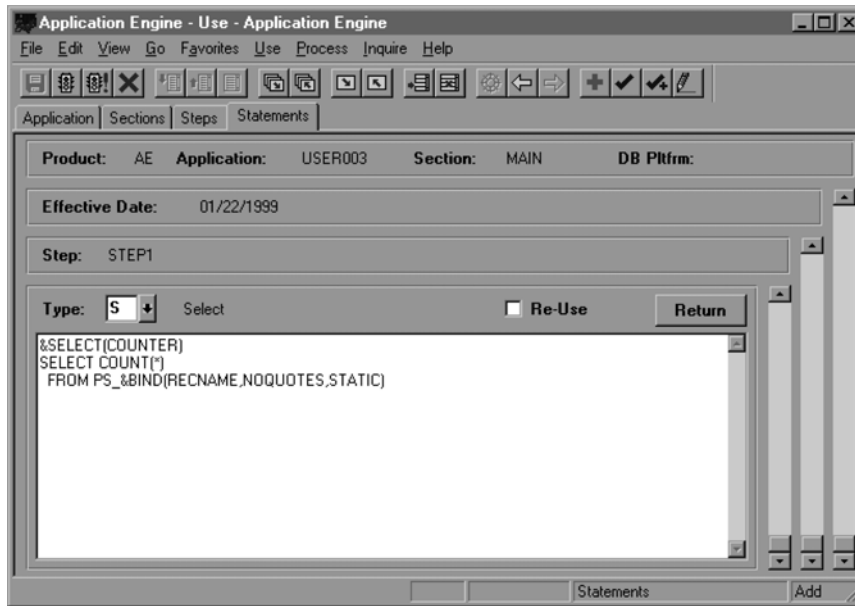


Figure 38.5 Entering the statement text

Notice the third line of the Select statement text:

```
&SELECT (COUNTER)

SELECT COUNT ( * )

FROM PS_&BIND (RECNAME , NOQUOTES , STATIC)
```

Remember, when we run this program through the Process Request panel we initialize the cache field RECNAME with the name of our table. In our test, we use JOB. Using the &BIND function Application Engine compiles the following SQL statement:

```
SELECT COUNT ( * )
FROM PS_JOB
```

Notice the RECNAME value JOB is prefixed by PS_. This is the standard PeopleSoft convention. The SQR version concatenates the PS_ with the entered RECNAME as well. Let's look at the &BIND value a little closer. RECNAME has a character datatype. If the NOQUOTES parameter was omitted, the resulting value would be JOB enclosed in quotation marks or 'JOB'. Since we are binding this value to the prefix PS_, the SQL would attempt to select from PS_ 'JOB', which is not valid and would cause an

error condition. The `STATIC` parameter tells Application Engine to resolve the `&BIND` value before the SQL statement is compiled.

The row count from the `JOB` table is stored in the cache field `COUNTER`.

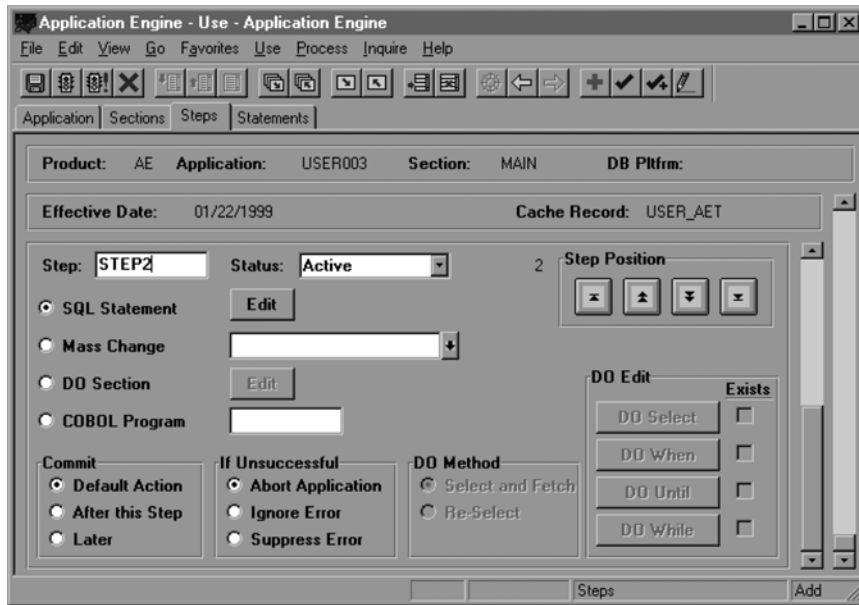


Figure 38.6 Defining STEP2

Figure 38.6 shows the completed Step Definition panel for STEP2. Let's move to the statement panel next.

38.1.3 Multiple &BIND parameters in a &MSG function

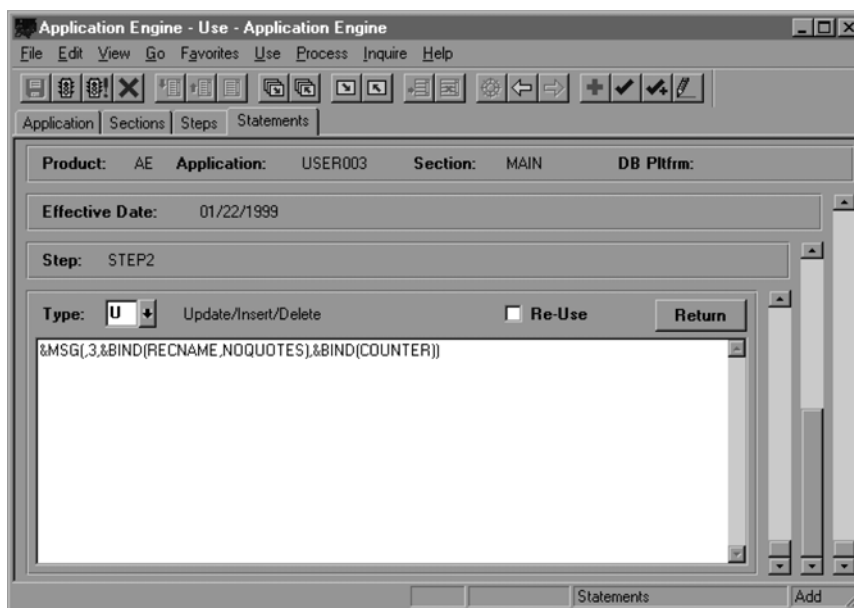


Figure 38.7 Entering &MSG statement text

Step 2 produces a message that includes two of our cache fields: the RECNAME initially entered by the user, and the COUNTER that's populated with the number of rows. We're using message number 3 of the default Message Set (20,001) which we entered as

%1 contains %2 records

Here is our &MSG function:

```
&MSG ( , 3 , &BIND ( RECNAME , NOQUOTES ) , &BIND ( COUNTER ) )
```

The RECNAME value will be inserted into the first parameter or %1 of the message text.

The COUNTER value will be inserted into the second parameter or %2 of the message text.

We can now test the USER003 Application using the Process Request panel.

38.1.4 Assign initial cache values on the Process Request panel

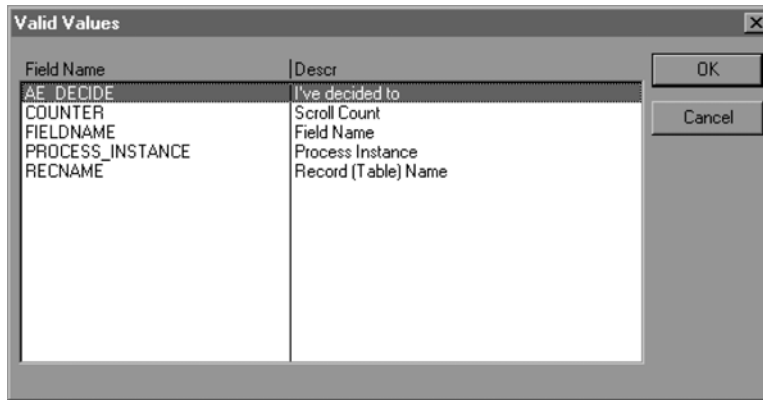
Assign a Run Control ID—#USER003 (figure 38.8).

Navigation: Go →PeopleTools →Application Engine →Process →Request →Request →Add

A dialog box titled "Add -- Request" with a close button (X) in the top right corner. It contains a text field labeled "Run Control ID:" with the value "#USER003" entered. To the right of the text field are two buttons: "OK" and "Cancel".

Figure 38.8
Assigning the
Run Control ID

Once we fill in the Product (AE) and Application (USER003), we can assign an initial value to any cache fields included in the cache record. Simply click on the cache field 'Down Arrow' to display the drop-down list box. Figure 38.9 shows the drop-down list with all of the cache fields on the USER_AET cache record. We included the field RECNAME when we built our cache record for the purpose of this exercise. Select the RECNAME cache field.

A dialog box titled "Valid Values" with a close button (X) in the top right corner. It contains a table with two columns: "Field Name" and "Descr". The table lists five cache fields: AE DECIDE, COUNTER, FIELDNAME, PROCESS_INSTANCE, and RECNAME, each with a corresponding description. To the right of the table are two buttons: "OK" and "Cancel".

Field Name	Descr
AE DECIDE	I've decided to
COUNTER	Scroll Count
FIELDNAME	Field Name
PROCESS_INSTANCE	Process Instance
RECNAME	Record (Table) Name

Figure 38.9 The cache field drop-down list box

Now assign the value JOB to our RECNAME cache field. Our application will substitute the value JOB in our Dynamic SQL Select statement.

Application Engine - Process - Request

File Edit View Go Favorites Use Process Inquire Help

Request Messages

Operator Id: PS Run Control ID: #USER003

Request Number: 1

Process Frequency: Always

Product: AE

Application: USER003 User Application 003

Fields

Field	Value
RECNAME	JOB

Request Update

Figure 38.10 Assigning an initial value to the cache field

Once again, highlight the AEADHOC process and click OK.

Process Scheduler Request

Operator ID: PS Run Control ID: #USER003

Run Location:
☒ Client ☐ Server
 Server:

Output Destination:
☒ File ☐ Printer ☐ Window
 File/Printer: %temp%*

Run Date/Time:
 Date: 01/22/99
 Time: 09:13:00 AM
 Reset to current Date/Time

Run Recurrence:
 Once
 Name: New Update Delete

OK Cancel

Description	Name	Process Type Descr
Application Engine	AEADHOC	Application Engine
Application Engine	PTPEMAIN	COBOL SQL

Figure 38.11 Submitting a Process Scheduler request

As the message log indicates, the USER003 program was successful. The Job record was dynamically called, and the correct number of rows were selected. Our totals match those generated by the USER003 SQR program (1685 rows).

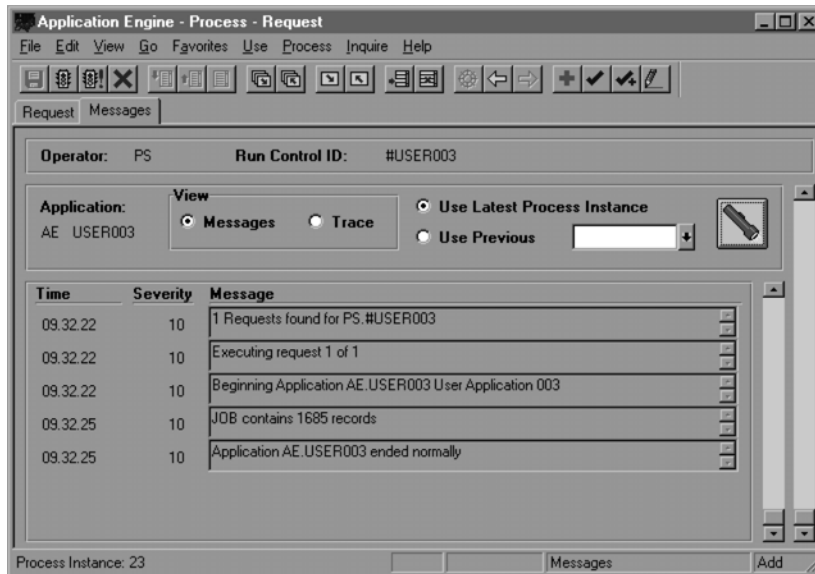


Figure 38.12 Reviewing Process Request messages

38.2 SQR/APPLICATION ENGINE COMPARISON

Now, let's look at the logical structure of both our programs:

SQR:

```
Begin-Program
  User prompted
  Main-Step1
  Main-Step2
```

Application Engine:

```
USER003
  Cache assignment
  MAIN.STEP1
  MAIN.STEP2
```

The structures are once again identical. The only exception lies in the method of entering the RECNAME. The SQR structure used an input prompt while the A/E program used the Process Request cache field assignment. Please note that a Run Control panel can be created for the SQR process, and values may be assigned in the same manner as the A/E program.

KEY POINTS

- 1 The &BIND function can create dynamic SQL statements.
- 2 The &BIND parameters NOQUOTES and STATIC are used to format the bind value within the SQL statement.
- 3 Multiple &BIND parameters are permitted in a &MSG function.
- 4 The Process Request panel allows us to assign initial cache field values. This allows us to test our application engine programs without having to build a Run Control panel.



CHAPTER 39

Selecting multiple rows

- 39.1 Exercise 4: Processing multiple rows 829
- 39.2 SQR/Application Engine comparison 848

In our last two exercises, we selected one row of data and displayed a message. We will now learn how to process multiple rows returned by our `Select`. Also, we have been using one section only—the `MAIN` section. We will introduce multiple sections in this next exercise. The additional sections will give us increased flexibility and control in our Application Engine program.

39.1 EXERCISE 4: PROCESSING MULTIPLE ROWS

In the last exercise, we selected and displayed the number of rows from a record entered by the user. We'll add a little bit more complexity in this next exercise. The user will now enter a FIELDNAME. We are going to select the number of rows in EACH record containing this field. A message will be generated for each record as well. We will access the PeopleTools table PSRECFIELD to determine which record(s) to select based on the field entered by the user.

39.1.1 Creating an SQR version

We begin this exercise by displaying the SQR version of this program:

Listing 39.1

USER004.sqr

```
! USER004.SQR

begin-program
input $fieldname 'Enter FIELDNAME' maxlen=18

do Main-Step1

end-program

begin-procedure Main-Step1

begin-select
a.recname

    let $recname      = &a.recname

    do Count-Step1
    do Count-Step2

from psrecfield      a,
    psrecdefn        b
where a.recname      = b.recname
    and a.fieldname = $fieldname
    and b.rectype    = 0
order by a.recname
end-select

end-procedure

begin-procedure Count-Step1

let $table    = 'ps_' || $recname
let #counter = 0
begin-select
```

```

count(*)          &counter
  let #counter = &counter
  from [$table]
end-select

end-procedure

begin-procedure Count-Step2

show ' '
show $recname ' Record Count: ' #counter

end-procedure

```

The user is prompted for a field name using the Input statement. Once again, we are not concerned with the validation of the user input. In our example, we use the field PAY_END_DT. Any record that has the PAY_END_DT field is selected and processed by the program.

Consider a portion of the SQR.LOG produced by the run:

```

Enter FIELDNAME: PAY_END_DT

BEN_PLAN_DATA Record Count: 0.000000
BOND_LOG Record Count: 471.000000
DED_CALC Record Count: 90.000000
DED_LINE Record Count: 29.000000
DED_MESSAGE Record Count: 0.000000
DED_WORK Record Count: 0.000000
ESPP_RUNCTL Record Count: 2.000000
GL_GEN_HISTORY Record Count: 0.000000
GP_CAL_BLD Record Count: 0.000000
GP_CRT_GER_AET Record Count: 0.000000
IMP_ADJUST Record Count: 0.000000
IMP_CALC Record Count: 0.000000
PAYROLL_ACCRUAL Record Count: 0.000000
PAY_CALC_RUNCTL Record Count: 0.000000
PAY_CALENDAR Record Count: 1170.000000
PAY_CALENDR_NLD Record Count: 241.000000
PAY_CAL_BAL_ID Record Count: 911.000000
PAY_CBLD_RUNCTL Record Count: 1.000000
PAY_CHECK Record Count: 7385.000000

Etc...

```

If you were to view each of these record definitions in Application Designer, you would see that each of the records listed above contains the field PAY_END_DT.

Now, let's duplicate this functionality in our Application Engine program.

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine →Application →Add

Figure 39.1
Naming the application

We'll begin by adding the USER004 Application name and the section MAIN. Set the description, cache record, version and message set number for our application (figure 39.2).

Figure 39.2 Defining the application

Now set the description on the section MAIN panel (figure 39.3).

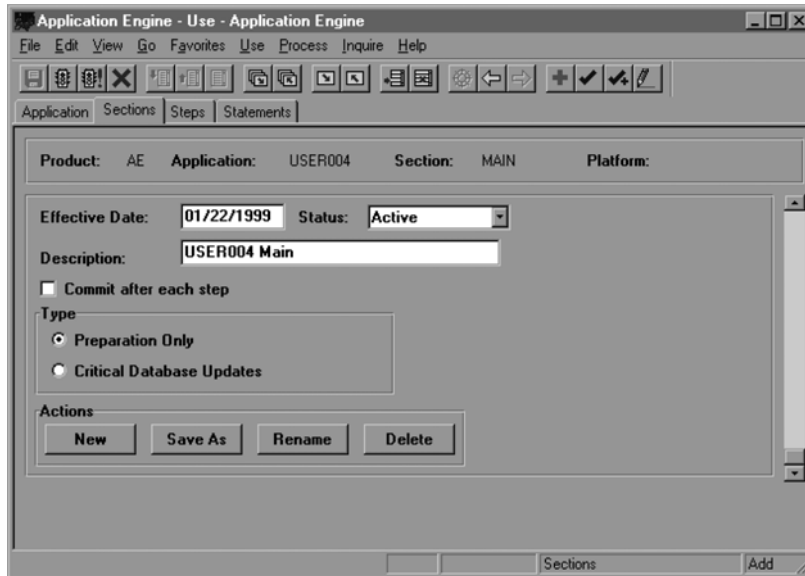


Figure 39.3 Defining section MAIN

We'll now proceed with STEP1. There hasn't been much variation in our exercises so far; this will change very soon.

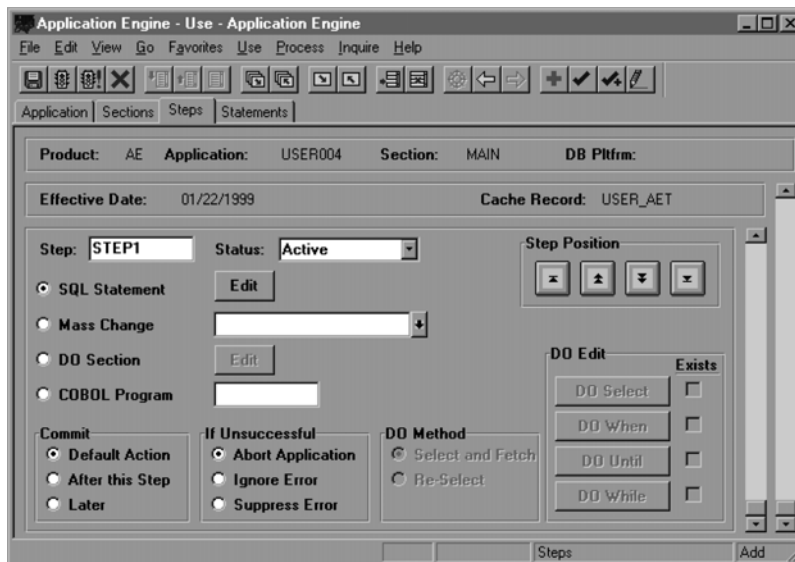


Figure 39.4 Defining STEP1

39.1.2 Using the DO Select statement type

In this exercise, we use a different statement type called a Do Select. For each row returned by the SQL Select, we perform or “DO” another section using the RECNAME value. Using the DO Select statement type tells Application Engine to call a “specified” subordinate section to process each returned row. Our new section uses this RECNAME value to dynamically select the number of rows, assign the number of rows to the cache field COUNTER, and once again display a message with the results.

Click on the Statements Folder tab. Figure 39.5 shows the statement type along with the SQL statement text we’re going to use.

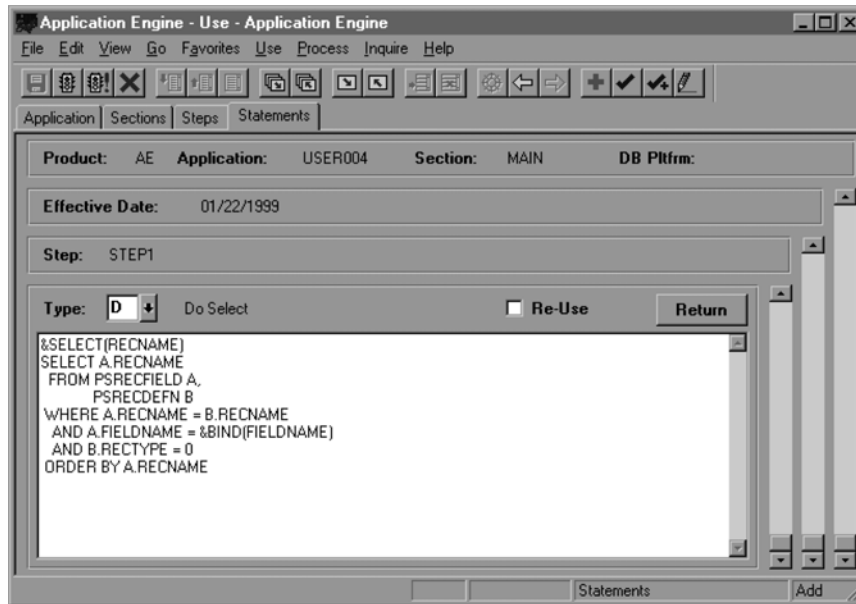


Figure 39.5 Entering DO Select statement text

Let’s examine our SQL statement closer:

```
&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
```

First of all, we are using two PeopleTools tables. PSRECFIELD stores all the record definitions created in Application Designer. We are selecting all RECNAME values that have the FIELDNAME value we've input on the Process Request screen. We're joining the PSRECFIELD table to another PeopleTools table called PSRECDEFN. This table is used to hold information about the record itself. The RECTYPE indicator is set to 0 if it is a physical SQL table. If it's not zero, it could be a type of view, work record or subrecord definition. Since the ultimate goal of our program is to determine the number of rows in each table, we need to make sure we're only selecting SQL table record definitions.

The &BIND function retrieves the cache field FIELDNAME. Remember, we're using the field PAY_END_DT in our example. We're using the FIELDNAME value as part of our selection criteria. The SQL criteria using &BIND is translated to:

```
AND A.FIELDNAME = 'PAY_END_DT'
```

Click back on the steps folder tab to return to the step definition panel (figure 39.6).

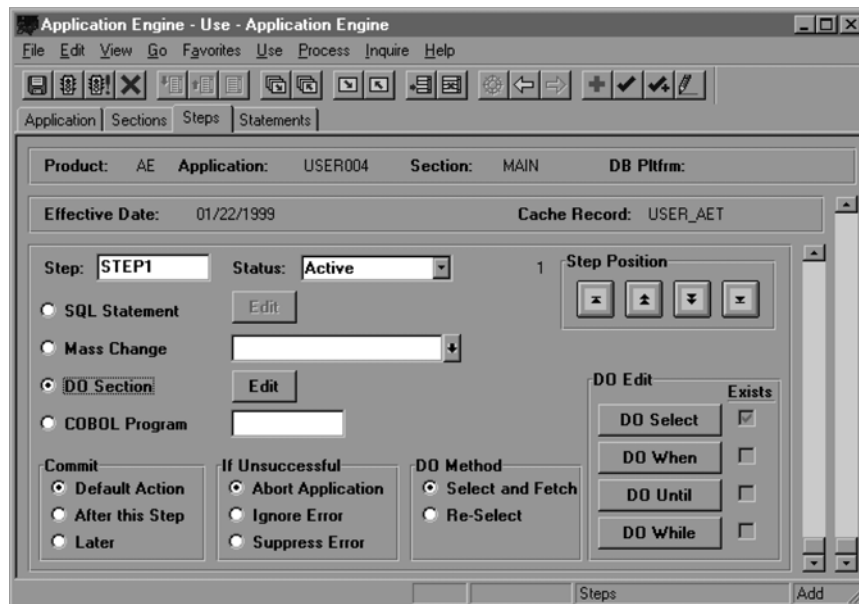


Figure 39.6 Using the DO section radio button

Notice the DO section radio button has been set. We controlled this by assigning the DO Select statement. Let's see what happens when we try to save our work (figure 39.7).

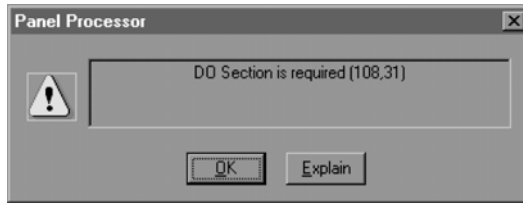


Figure 39.7
Error message—
DO section is required

An error message is produced. We click on the Explain button for further information (figure 39.8).

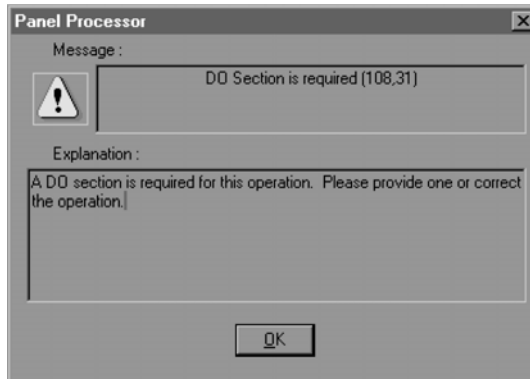


Figure 39.8
Same error message with explanation

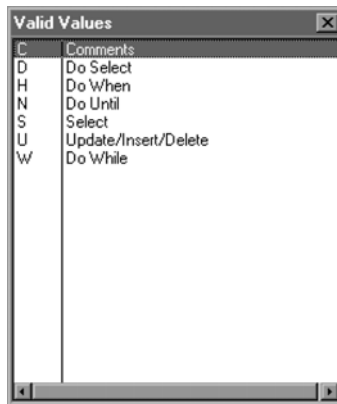


Figure 39.9 Statement type
drop-down list box

Because we've used a statement type of DO Select, a DO section is required. This presents a problem because we haven't created the new section yet. We have to temporarily set the radio button back to SQL statement instead of Do Section. We can also set the statement type to Comments by clicking on the Edit push button (next to SQL statement) or by clicking on the Statements folder tab. The Statement panel has a drop-down box for statement types. Let's set the statement type to Comments for now.

You can use this statement type to add descriptions to your program. In our case, we use it to alleviate the problem we're having. We can now save our record. Once we create our new section, we can go back and change the statement type to Do Select. At that point, we'll link our new section to our program.

Please note that PeopleSoft will allow you to save the record without setting the statement type to Comments. In this particular case, it's a good practice to de-activate the DO Select since a DO section is not yet attached. Let's create the new section now.

39.1.3 Creating and using additional sections

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine →Section →Add

Figure 39.10
Adding a new section

When the Add edit box appears, we use the same product and application (figure 39.10). This is the first time we are adding a section name. Up until now, the first section has always been filled in for us. Remember the first section is always MAIN. We call our new section COUNT. It's an appropriate name since we'll determine the count of each REC-NAME passed to this section and display the results to the message log.

On the section definition screen, we enter a simple description.

Figure 39.11 Defining section COUNT

Since we are in a new section, we can once again call our step STEP1. After all, this is the first step of the COUNT section. Now let's move to the statement definition panel.

Application Engine - Use - Application Engine

File Edit View Go Favorites Use Process Inquire Help

Application Sections Steps Statements

Product: AE Application: USER004 Section: COUNT DB Pltfrm:

Effective Date: 01/22/1999 Cache Record: USER_AET

Step: STEP1 Status: Active

SQL Statement ☒ Edit

Mass Change ☐ +

DO Section ☐ Edit

COBOL Program ☐

Commit

☒ Default Action

☐ After this Step

☐ Later

If Unsuccessful

☒ Abort Application

☐ Ignore Error

☐ Suppress Error

DO Method

☒ Select and Fetch

☐ Re-Select

DO Edit

☒ DO Select

☐ DO When

☐ DO Until

☐ DO While

Exists

☐

☐

☐

☐

Steps Add

Figure 39.12 Defining STEP1 of COUNT section

Application Engine - Use - Application Engine

File Edit View Go Favorites Use Process Inquire Help

Application Sections Steps Statements

Product: AE Application: USER004 Section: COUNT DB Pltfrm:

Effective Date: 01/22/1999

Step: STEP1

Type: S Select

☐ Re-Use

Return

&SELECT(COUNTER)
SELECT COUNT(*)
FROM PS_&BIND(RECNAME,NOQUOTES,STATIC)

Statements Add

Figure 39.13 Adding Select statement text

You may have noticed this statement is identical to STEP1 in exercise 3. The reason is simple: We have a table determined by retrieving the value in the cache field RECNAME. The purpose of this step is to select the number of rows from that table. That purpose hasn't changed. In exercise 3, the cache field RECNAME was assigned through the Process Request Panel. In this exercise, the RECNAME value is assigned by the calling step MAIN.STEP1. We'll demonstrate the reuseability of applications/sections at the end of this exercise.

We'll now enter the second step of the COUNT section which produces our message log entry (figure 39.15).

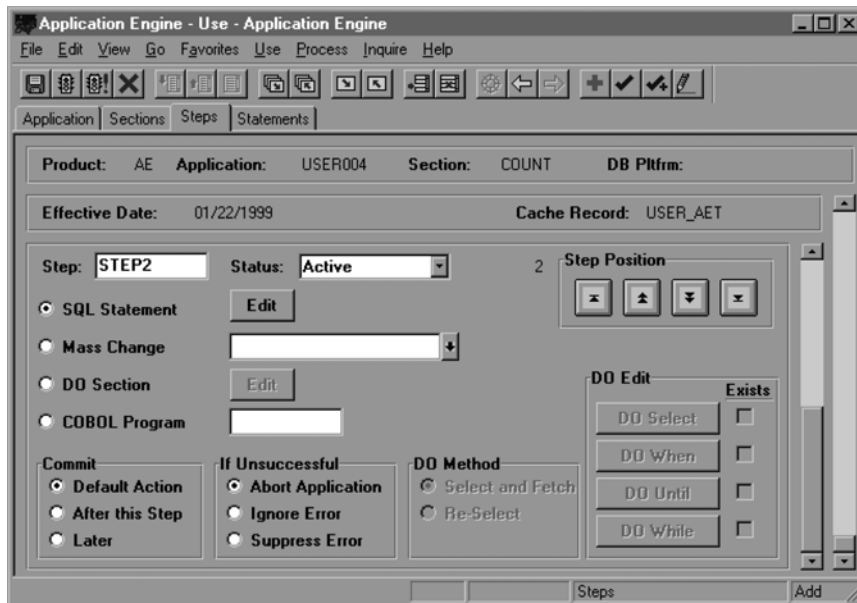


Figure 39.14 Defining STEP2 of section COUNT

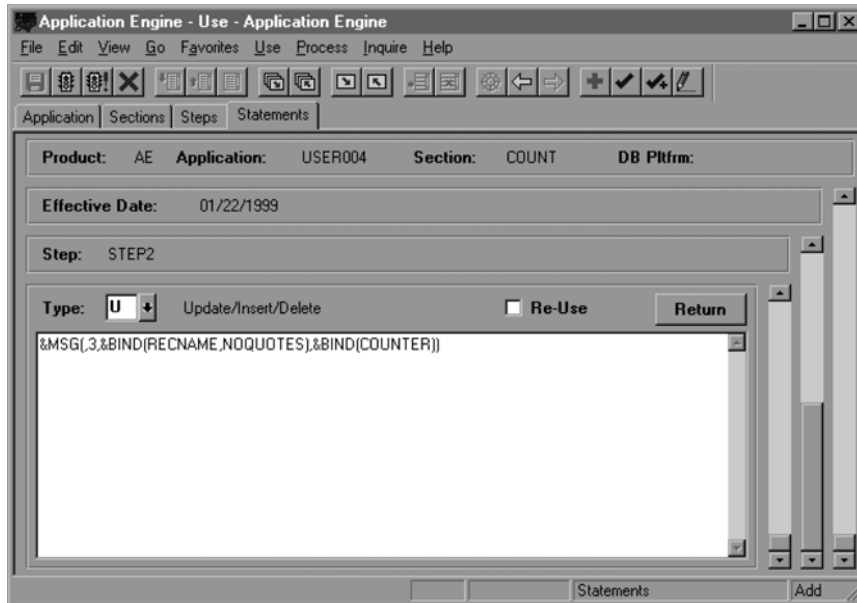


Figure 39.15 Adding &MSG statement text

As in step1, the STEP2 portion is identical to that of our prior exercise. Once we save the new section, we're ready to link it to our MAIN section.

We go back to the MAIN section using Correction Mode. Enter the product we're using (PS/AE), and click on the Search push button. You can see all of the applications and sections we've defined in our exercises. Select the MAIN section of the USER004 application (figure 39.16). Once we link our new section, we are ready to test.

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine →Application →Correction

Section	Descr	DB Pltfrm	Appl ID
COUNT	USER004 Count		USER004
MAIN	USER001 Main		USER001
MAIN	USER002 Main		USER002
MAIN	USER003 Main		USER003
MAIN	USER004 Main		USER004

Figure 39.16 Application/section list box for all of our PS/AE exercises to date

Product: AE Application: USER004 Section: MAIN DB Pltfrm:

Effective Date: 01/22/1999 Cache Record: USER_AET

Step: STEP1 Status: Active 1 Step Position

☐ SQL Statement
☐ Mass Change
☒ DO Section
☐ COBOL Program

Commit: ☒ Default Action ☐ After this Step ☐ Later
 If Unsuccessful: ☐ Abort Application ☐ Ignore Error ☐ Suppress Error
 DO Method: ☒ Select and Fetch ☐ Re-Select

DO Edit Exists

☐
 ☐
 ☐
 ☐

Steps Correct

Figure 39.17 Resetting the DO section radio button

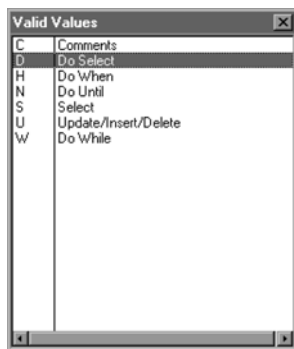


Figure 39.18 Statement type drop-down list box

Return to the statement definition by clicking on the Statements folder tab. Once we change our statement type back to DO Select (figure 39.18), we can proceed with linking our new section COUNT. Remember we deactivated the step by assigning a Comment statement type.

The DO section radio button should now be set. Click on the Edit button to link our new section.

Set the product and application and then click on the section drop-down box.

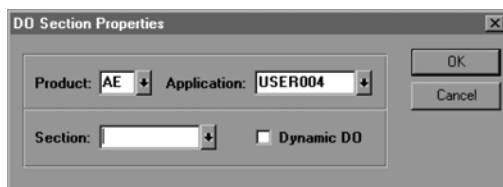


Figure 39.19 DO Section Properties dialog box

You'll see the valid choices for the AE.USER004 program. Our new section COUNT is in the drop-down list. Select the COUNT section (figure 39.20).

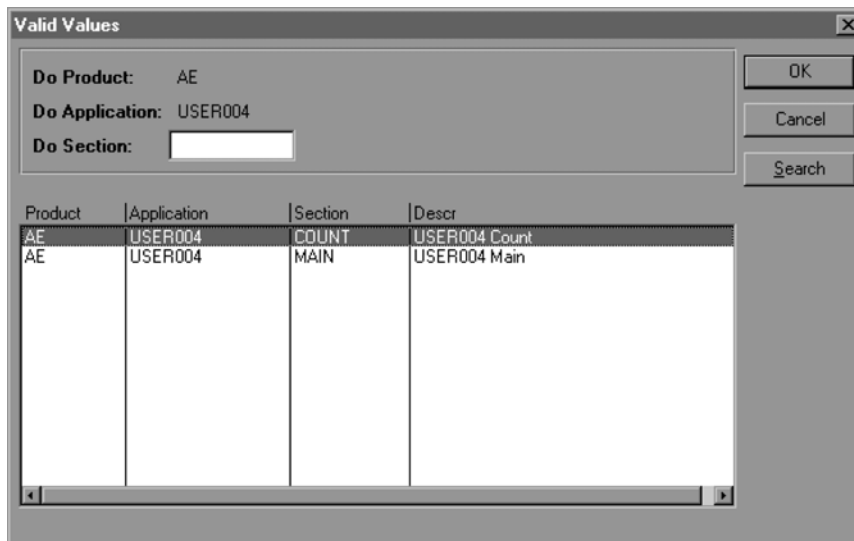


Figure 39.20 Selecting the DO section

Once selected, we return to the DO Section Properties box with our section COUNT filled in. Click the OK button.

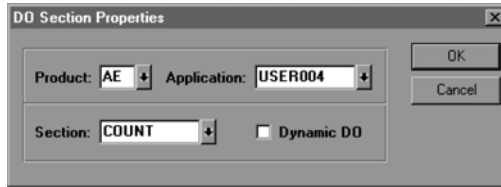


Figure 39.21
The completed DO Section Properties dialog box

When we return to the Step Definition screen, we see the COUNT section displayed to the right of the DO section radio button. For each row returned by the STEP1 DO Select statement, the COUNT section will be executed. Save your work. Let's test our new program.

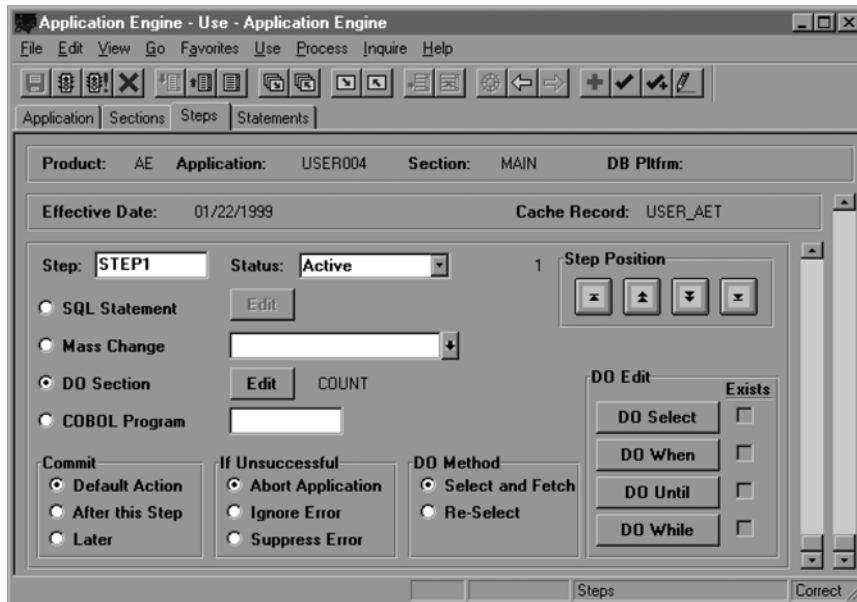


Figure 39.22 The DO Section has been completed

Return to the Process Request Panel and add the Run Control ID #USER004 (figure 39.23). Now, click the OK button.

Navigation: Go →PeopleTools →Application Engine →Process →Request →Request →Add

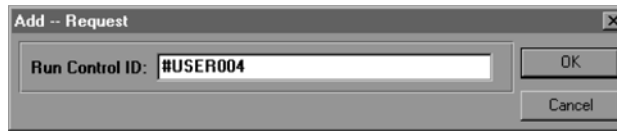


Figure 39.23
Assign the Run Control ID

When the Process Request panel appears, click on the Fields edit box and highlight the FIELDNAME cache field. Click OK to proceed (figure 39.24).

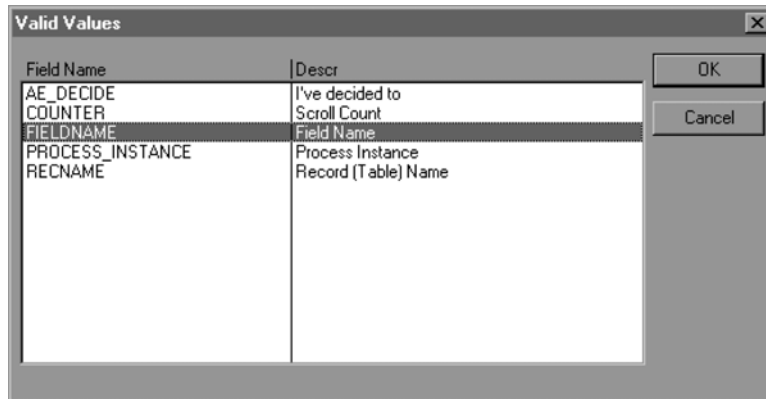


Figure 39.24 The cache field drop-down list box

We assign the value `PAY_END_DT` to our cache field (figure 39.25). Our Application Engine program now displays the number of rows in every table that has the field `PAY_END_DT`.

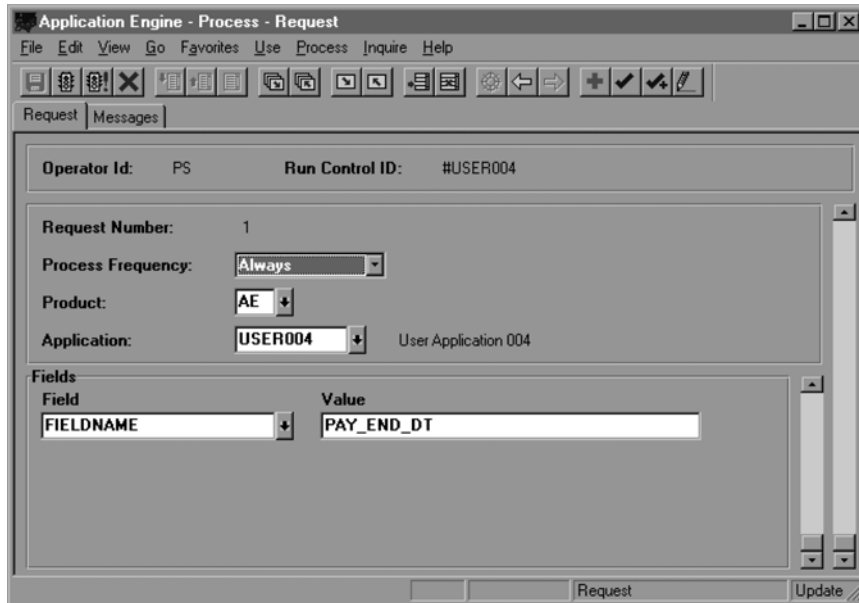


Figure 39.25 Assigning an initial value to the cache field

Once again, highlight the AEADHOC process and click OK.

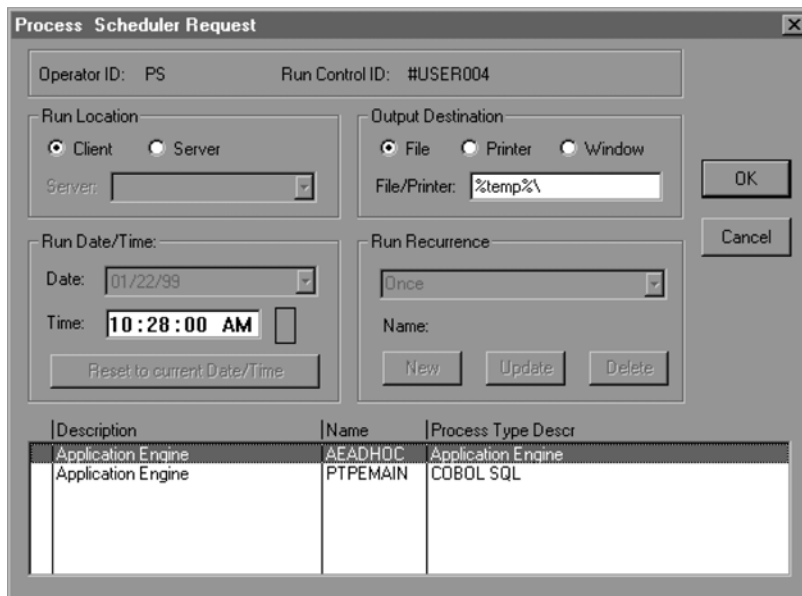


Figure 39.26 Submitting a Process Scheduler request

When running on the client, you may notice an MS-DOS box appear (figure 39.27). You may have missed it in earlier exercises since they processed much quicker. This screen shows the steps as they execute along with any messages generated. You can see our process seems to be working. The records are being displayed with the row counts. When the process ends, we'll look at the Message Log panel.

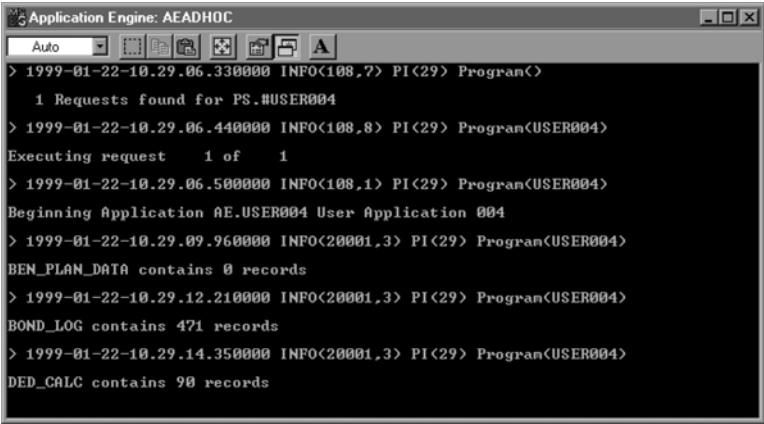


Figure 39.27 AEADHOC MS-DOS box

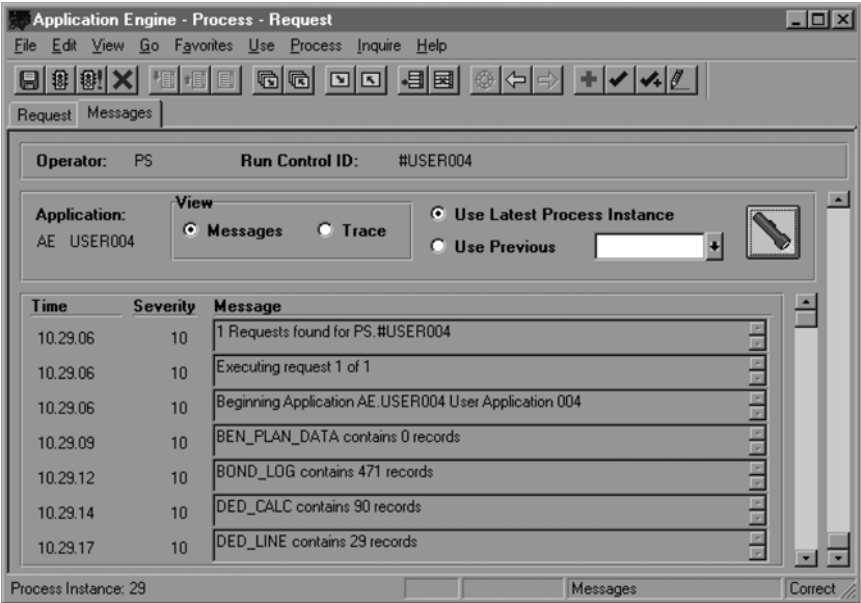


Figure 39.28 Reviewing Process Request messages

Another successful run! The message log output matches that of our SQR version. Before we end this exercise, let's talk about reuseability. During the creation of the COUNT section, we noticed it was identical to the USER003 program we created in exercise 3. Instead of creating a new section, we could routinely have called the USER003.MAIN section to accomplish the same task.

Let's give it a try.

39.1.4 Section reusability

Return to the Step Definition panel in our MAIN Section for STEP1. Click on the Edit button next to the DO section radio button. Instead of linking to Application USER004 section COUNT, link to Application USER003 Section MAIN (figure 39.29). Click the OK button.

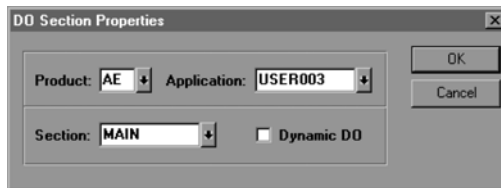


Figure 39.29
DO Section Properties reassignment

We are now linked to a different DO section (figure 39.30). AE.USER003.MAIN will now be performed instead of COUNT. Notice the fully qualified section name indicating product, application, and section. This allows you to borrow routines from other Application Engine programs. Let's test our changes.

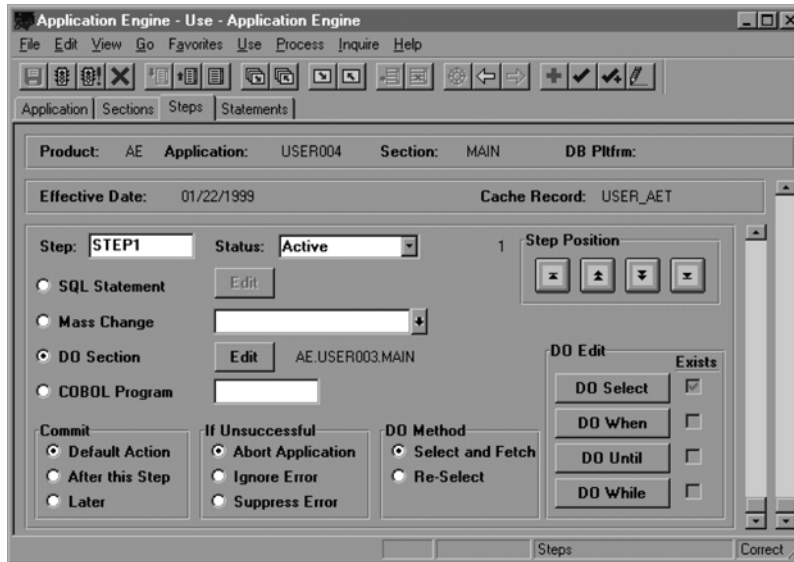


Figure 39.30 Revised DO section

Use the Process Request panel to execute the program. We can now examine the results on the Messages panel (figure 39.31).

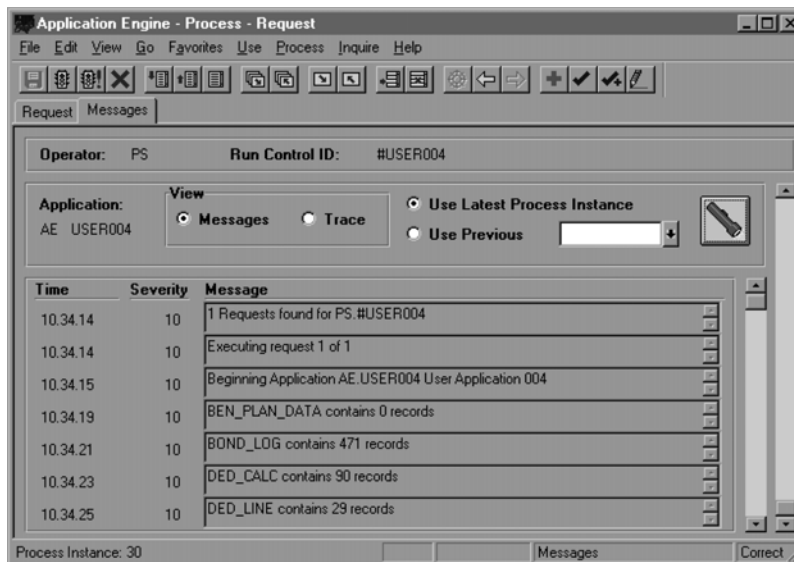


Figure 39.31 Review Process Request messages

Our results are the same as before. This demonstrates the reuseability of Application Engine sections.

39.2 ***SQR/APPLICATION ENGINE COMPARISON***

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

SQR:

```
Begin-Program
  User prompted
  Main-Step1
    Count-Step1
    Count-Step2
```

Application Engine:

```
USER003
  Cache assignment
  MAIN.STEP1
    COUNT.STEP1
    COUNT.STEP2
```

The structure is identical. The Main Step selects the record names that have the field name entered by the user. For each record selected, the COUNT section is processed. Step 1 selects the number of rows, and step 2 generates a message showing the results.

KEY POINTS

- 1 Multiple Rows may be processed one at a time using a DO Select statement.
- 2 Create and use additional sections to process each row.
- 3 Application Engine sections are reusable. This means a section that exists in one application engine program can be called from another. Sections which perform common tasks can be created and used by multiple programs.



CHAPTER 40

Incorporating decision logic

40.1 Exercise 5: Only process tables with rows 849

40.2 SQR/Application Engine comparison 870

In any programming language, the most vital function is the ability to make decisions and act accordingly. The purpose of this next chapter is to demonstrate the decision-making capability within Application Engine.

40.1 EXERCISE 5: ONLY PROCESS TABLES WITH ROWS

Exercise 4 selected a group of records, determined the number of rows in each, and displayed a message with the results. If you look at the message log, you'll notice many tables have zero rows. To demonstrate the decision-making capability of Application Engine, we're going to produce the messages only for tables that have rows of data.

40.1.1 Creating an SQR version

We'll begin this exercise by displaying the SQR version of this program:

Listing 40.1

USER005.sqr

```
! USER005.SQR

begin-program

input $fieldname 'Enter FIELDNAME' maxlen=15

do Main-Step1

end-program

begin-procedure Main-Step1

begin-select

a.recname

    let $recname      = &a.recname

    do Count-Step1

    from psrecfield    a,
        psrecdefn     b
where a.recname      = b.recname
    and a.fieldname   = $fieldname
    and b.rectype     = 0
order by a.recname

end-select

end-procedure

begin-procedure Count-Step1

let $stable   = 'ps_' || $recname
let #counter = 0

begin-select

count(*)      &counter

    let #counter = &counter

    if #counter > 0
        do Msg-Step1
    end-if
```



```

from [$table]

end-select

end-procedure

begin-procedure Msg-Step1

show ' '
show $recname ' Record Count: ' #counter

end-procedure

```

Once more the user is prompted for a fieldname using the Input statement again. For each table containing the fieldname, a record count is determined. If the record count is greater than zero, a message is produced.

Below is a portion of the SQR.log produced by the run:

```

Enter FIELDNAME: PAY_END_DT

BOND_LOG Record Count: 471.000000
DED_CALC Record Count: 90.000000
DED_LINE Record Count: 29.000000
ESPP_RUNCTL Record Count: 2.000000
PAY_CALENDAR Record Count: 1170.000000
PAY_CALENDR_NLD Record Count: 241.000000
PAY_CAL_BAL_ID Record Count: 911.000000
PAY_CBLD_RUNCTL Record Count: 1.000000
PAY_CHECK Record Count: 7385.000000
PAY_DEDUCTION Record Count: 45710.000000
PAY_DISTRIBUTN Record Count: 975.000000
PAY_EARNINGS Record Count: 13943.000000
PAY_GARNISH Record Count: 262.000000
PAY_GARN_OVRD Record Count: 2.000000
PAY_INS_EARNS Record Count: 8331.000000
PAY_LINE Record Count: 5104.000000
PAY_MESSAGE Record Count: 1.000000

Etc...

```

Notice, no tables are displayed with zero rows.

Now, let's duplicate this functionality in our Application Engine program.

We begin by adding the USER005 application name and the section MAIN (figure 40.1).

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine →Application →Add

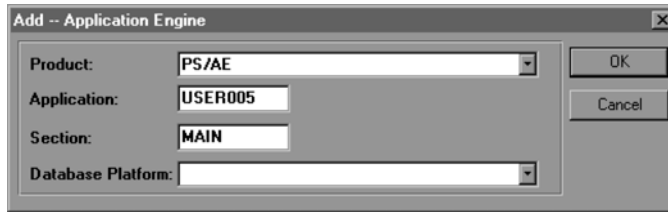


Figure 40.1
Naming the application

Set the description, cache record, version and message set number for our application (figure 40.2).

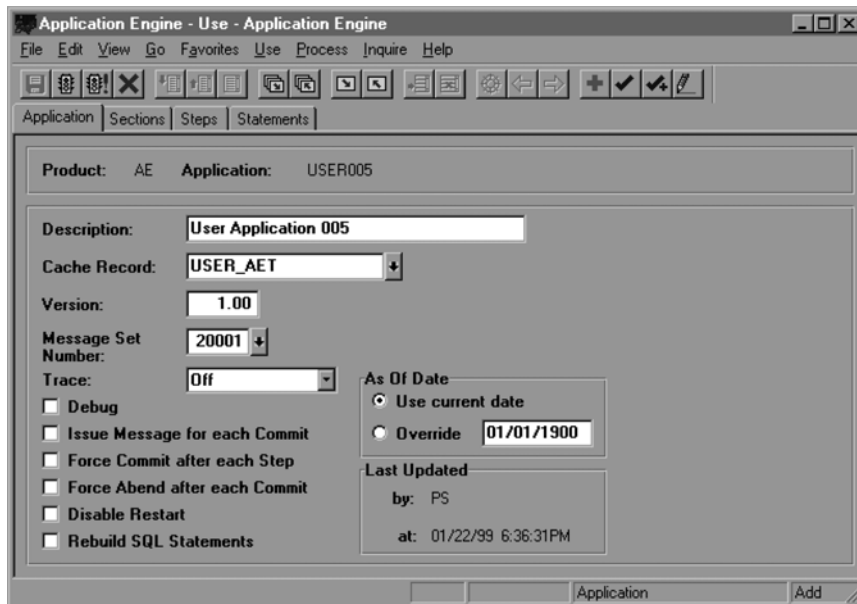


Figure 40.2 Defining the application

Now, set the description on the section MAIN panel.

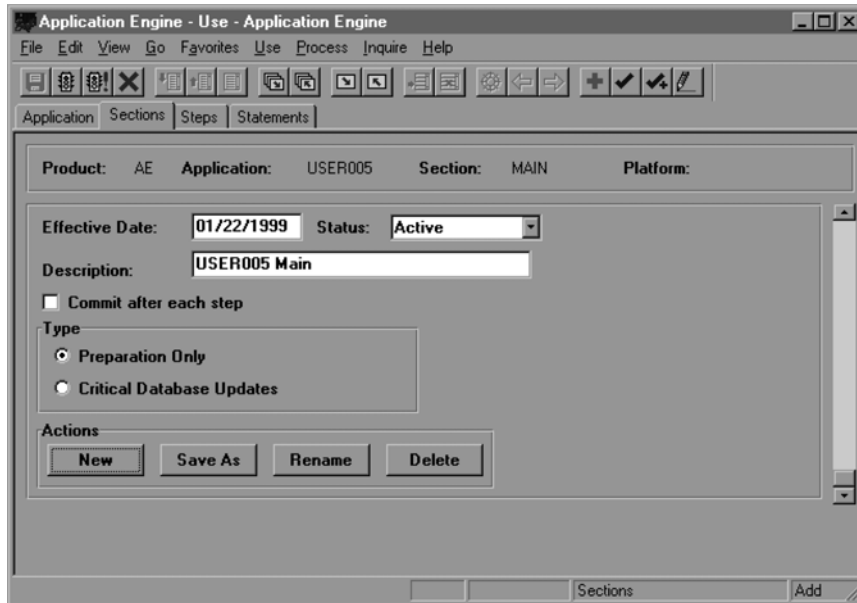


Figure 40.3 Defining section MAIN

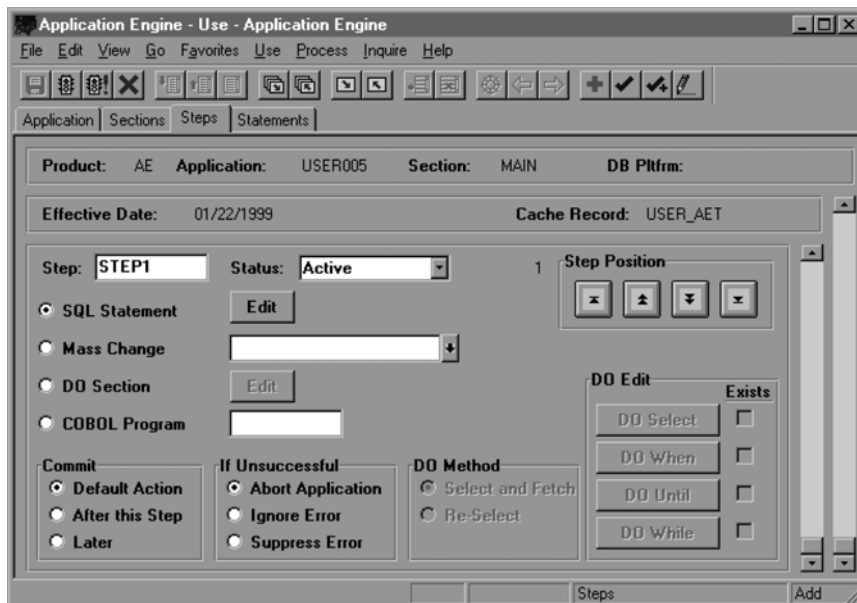


Figure 40.4 Defining STEP1

Step1 of our MAIN section in this exercise is identical to Step1 of the MAIN section of exercise 4. We did learn a small lesson in the last exercise. When using a DO Select statement type, a DO section needs to be defined in order to save the record. We need to create the new section. For now, let's change the statement type to Comment (figure 40.5). Now, save your work.

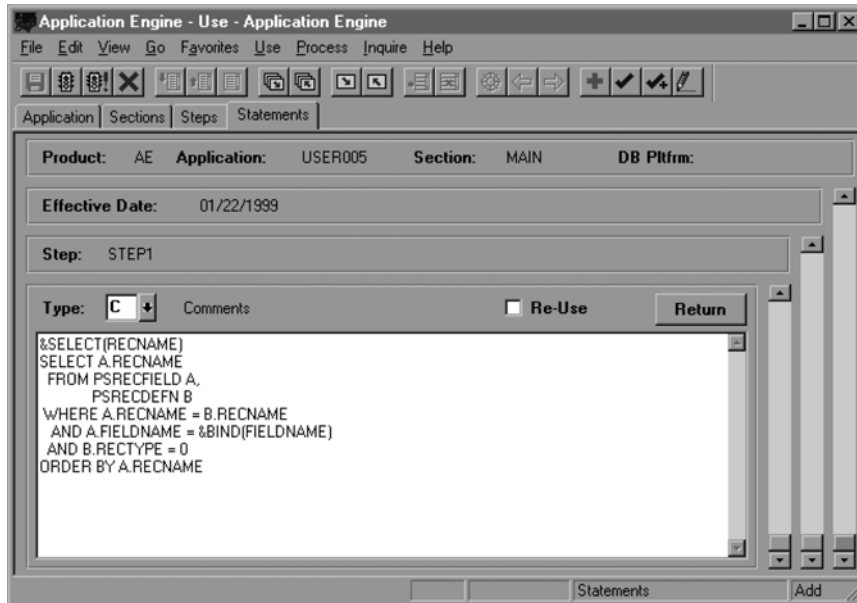


Figure 40.5 DO Select statement with Comment statement type

NOTE The MAIN section does not have to be created first. If you carefully plan the structure of your program before you actually begin building it, you can start with the subordinate sections and work your way backward. We'll demonstrate this approach in exercise #7 found in chapter 42 (Using Run Controls).

Once again, we add a section called COUNT to our application (figure 40.6).

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine →Section →Add

Figure 40.6
Adding a new section

Enter a brief description for the section COUNT.

Figure 40.7 Defining section COUNT

We call the first step in our COUNT section STEP1.

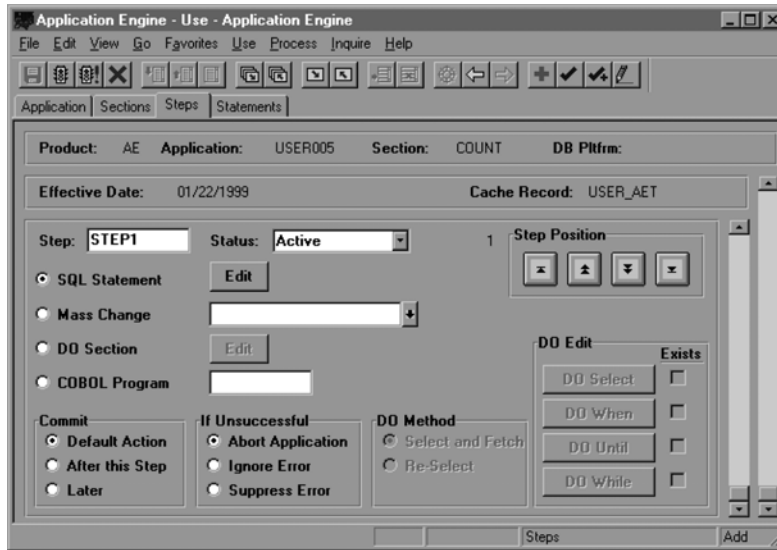


Figure 40.8 Defining STEP1 of section COUNT

Our `Select` statement hasn't changed since our last exercise. The cache field `COUNTER` is assigned the number of rows in the table. We'll revisit this after we create another section to display a message.

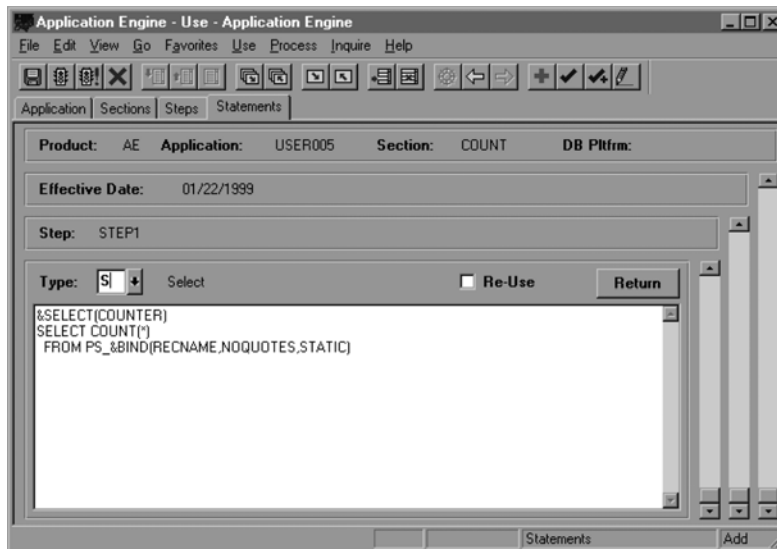


Figure 40.9 Adding Select statement text

We create a new section called MSG. The purpose of this routine is simply to display the table and number of rows in the table (figure 40.10).

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine →Section →Add

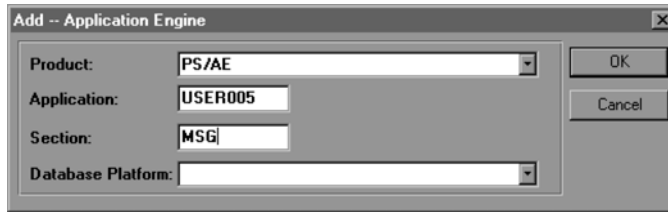
A dialog box titled "Add -- Application Engine" with a close button (X) in the top right corner. It contains four input fields: "Product:" with a dropdown menu showing "PS/AE", "Application:" with a text box containing "USER005", "Section:" with a text box containing "MSG", and "Database Platform:" with a dropdown menu. To the right of these fields are two buttons: "OK" and "Cancel".

Figure 40.10
Adding another section

Once again a brief description would be appropriate (figure 40.11).

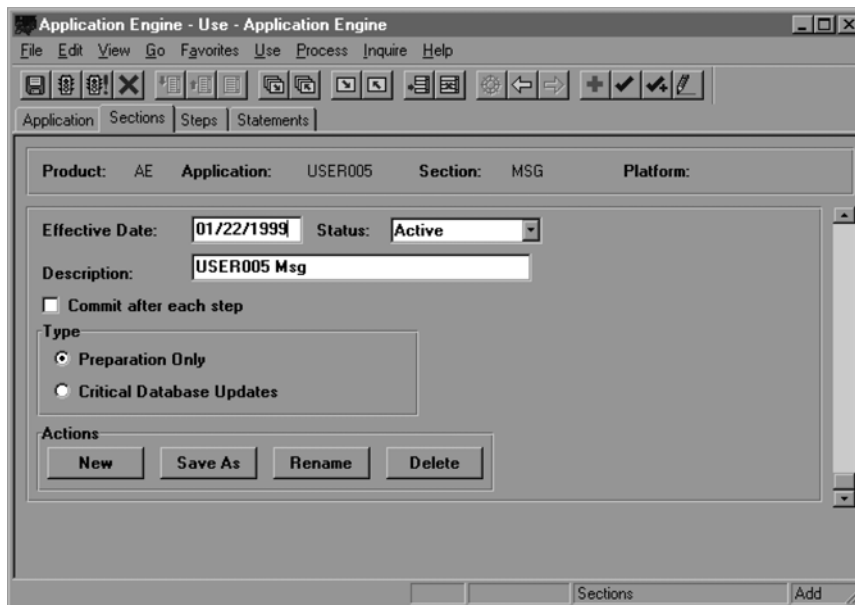
A screenshot of the "Application Engine - Use - Application Engine" window. The title bar includes standard window controls. Below the title bar is a menu bar with "File", "Edit", "View", "Go", "Favorites", "Use", "Process", "Inquire", and "Help". Below the menu bar is a toolbar with various icons. Below the toolbar are four tabs: "Application", "Sections", "Steps", and "Statements", with "Sections" currently selected. The main area displays the following information: "Product: AE", "Application: USER005", "Section: MSG", and "Platform:". Below this, there are two input fields: "Effective Date:" with a date picker showing "01/22/1999" and "Status:" with a dropdown menu showing "Active". Below these is a text box for "Description:" containing "USER005 Msg". There is a checkbox labeled "Commit after each step" which is currently unchecked. Below the checkbox is a section labeled "Type" with two radio buttons: "Preparation Only" (which is selected) and "Critical Database Updates". At the bottom of the main area is a section labeled "Actions" with four buttons: "New", "Save As", "Rename", and "Delete". At the bottom right of the window, there is a status bar with the text "Sections" and an "Add" button.

Figure 40.11 Defining section MSG

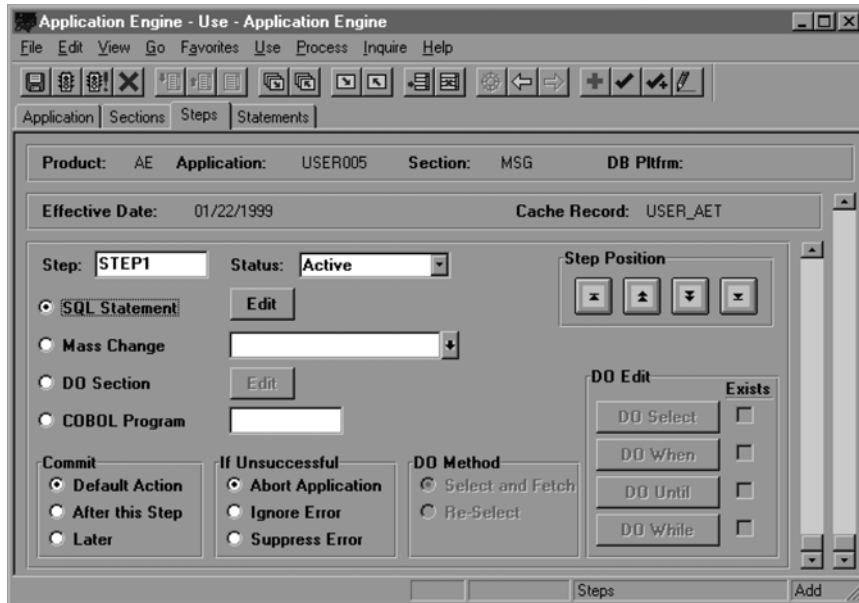


Figure 40.12 Defining STEP1 of section MSG

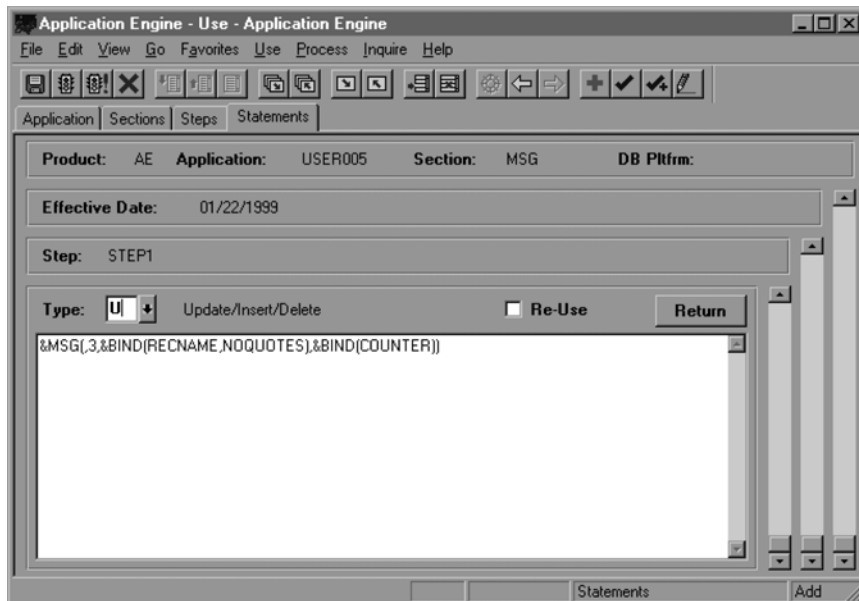


Figure 40.13 Adding the &MSG statement text

The &MSG function hasn't changed from the last exercise either. One main difference exists between this exercise and the last. In exercise 4, the message was included in a step immediately following the first step. The message step was executed unconditionally. In this exercise, we've place the message step in an entirely new section called MSG. The MSG section will only be performed if the row count is greater than zero. We now return to our MAIN section to link the COUNT section to our application (figure 40.14).

NOTE Remember the &MSG function only works if the statement type is set to "U".

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine →Application →Correction

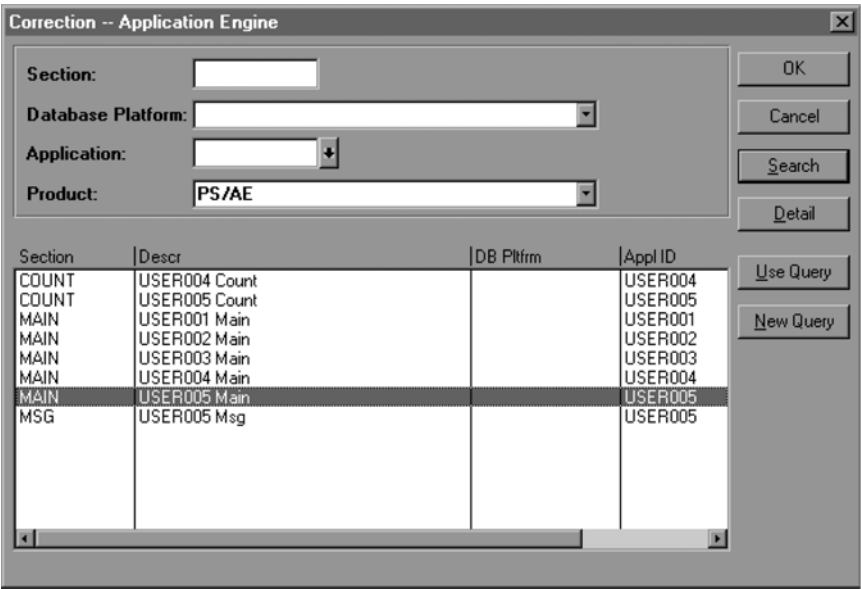


Figure 40.14 Application/section list box for all of our PS/AE exercises to date

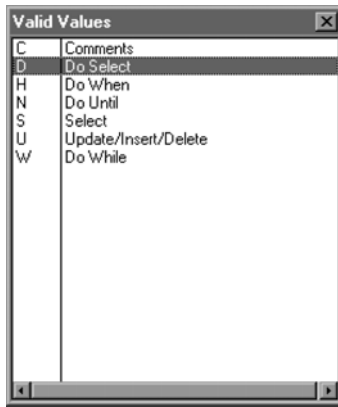


Figure 40.15 Statement type drop-down list box

Return to the MAIN section of our application (USER005) in Correction mode. Remember to enter the product we're using (PS/AE), and click on the Search push button. You'll see all of the applications and sections we've defined in our exercises. Select the MAIN section of the USER005 application.

Return to the statement definition by clicking on the Statements folder tab. Once we change our statement type back to DO Select, we can proceed with linking our section COUNT (figure 40.15).

The SQL statement type is set to DO Select. Now click on the DO section edit button (figure 40.16).

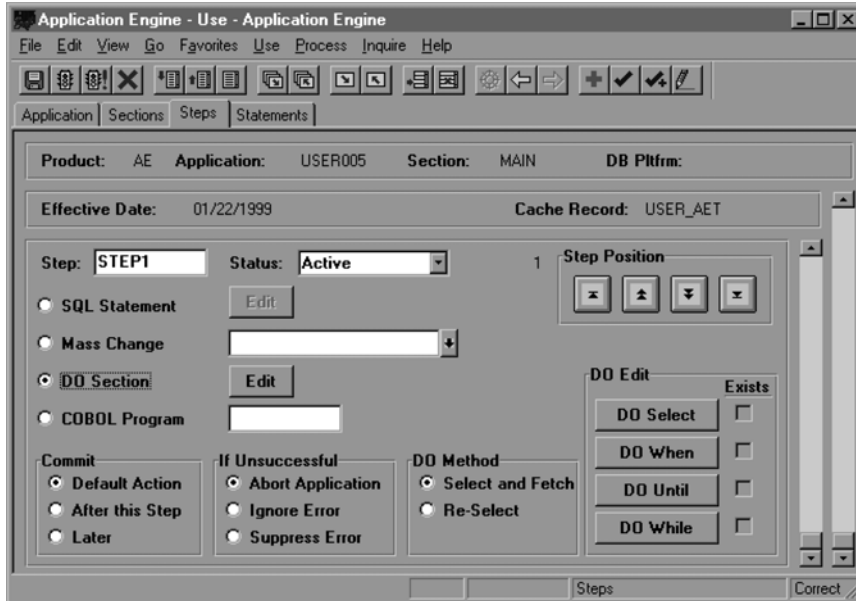


Figure 40.16 Setting the DO section radio button

Set the product and application and then click on the section drop-down box (figure 40.17).

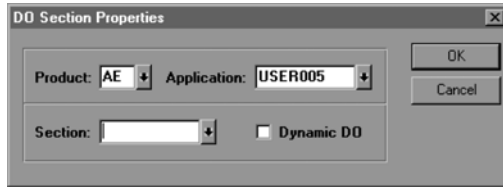


Figure 40.17
DO Section Properties dialog box

Along with our MAIN section, you can see the other two sections we've created. We need to link the COUNT section to our MAIN Select. Click on the COUNT section.

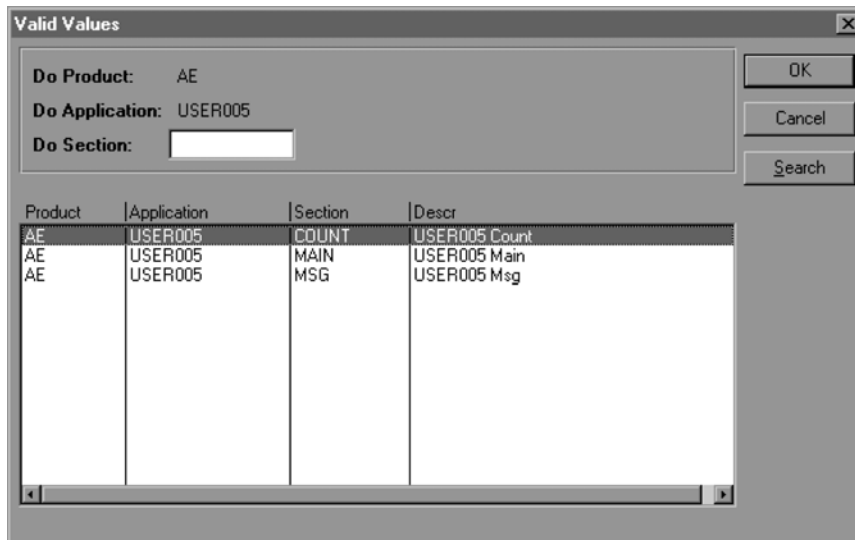


Figure 40.18 Selecting the DO section

Once selected, we return to the DO Section Properties Box with our section COUNT filled in. Click the OK button.

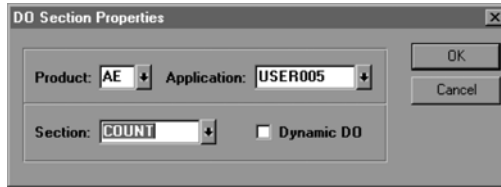


Figure 40.19
The completed DO Section dialog box

When we return to the Step Definition screen we see the COUNT section displayed to the right of the DO section radio button (figure 40.20). For each row returned by the STEP1 DO Select statement the COUNT section will be executed. We'll now return to the COUNT section.

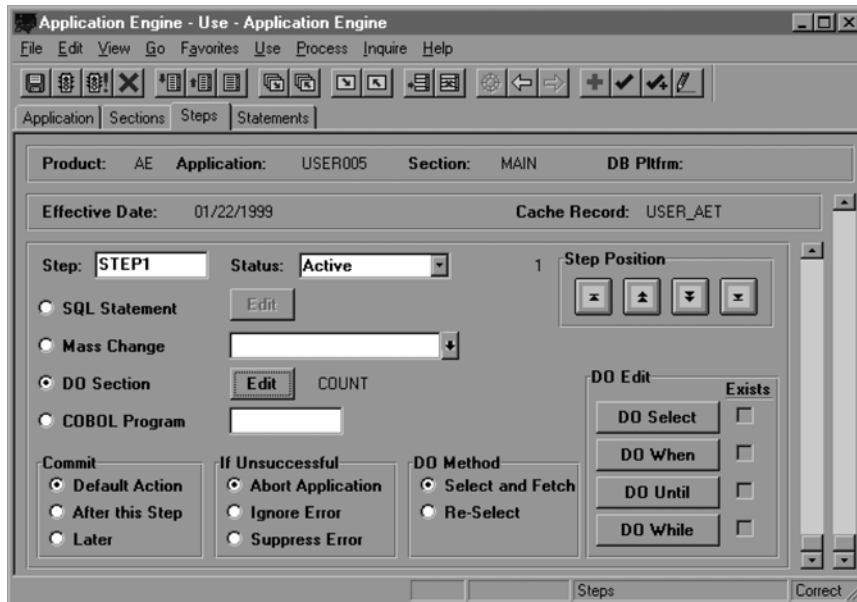


Figure 40.20 The DO Section has been completed

Return to the COUNT Section of our Application (USER005) in Correction Mode. Once again set the Product to PS/AE and click on the Search push button. Select the COUNT section of our USER005 application.

Navigation: Go →PeopleTools →Application Engine →Use →Application Engine →Application →Correction

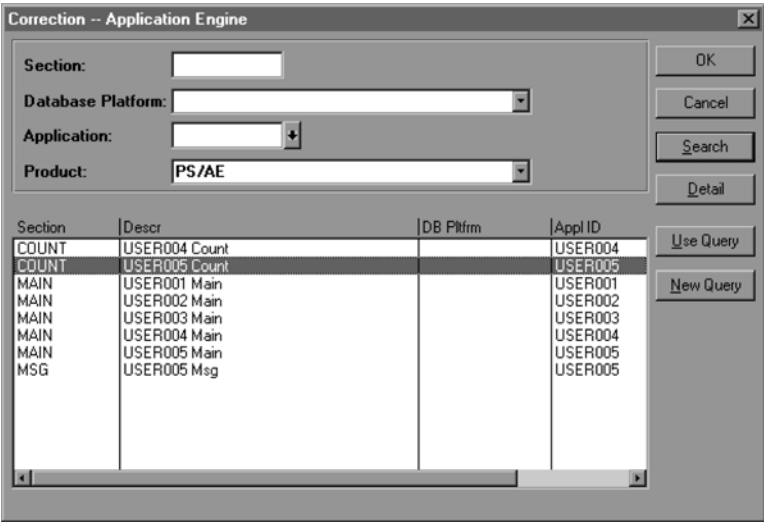


Figure 40.21 Application/section list box again

Add a second step called STEP2 to our COUNT section (figure 40.22). You can use the F7 Key to insert a new row.

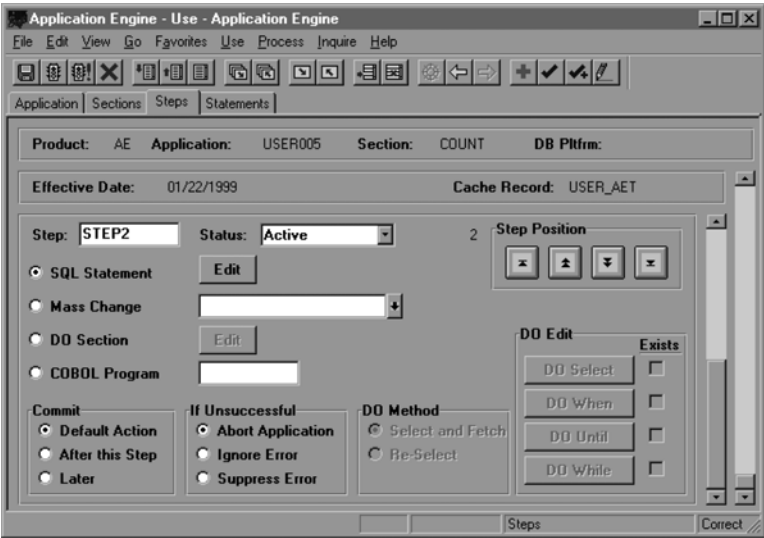


Figure 40.22 Creating STEP2 in section COUNT

40.1.2 Introducing the DO When statement type

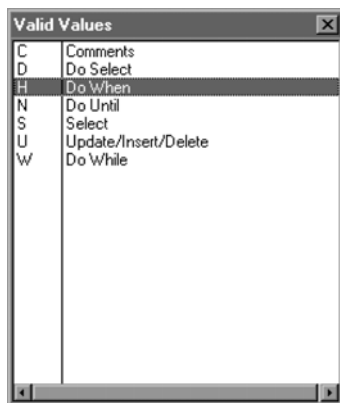


Figure 40.23 Select the DO When statement type

Use the Statements folder tab to define the statement. Click on the Statement Type edit box.

Select the DO When statement type in the drop-down list box shown in figure 40.23.

We're using a new statement type called DO When (figure 40.24). A DO section will be performed based on the results of a True or False Select statement. In our case we are evaluating the cache field COUNTER.

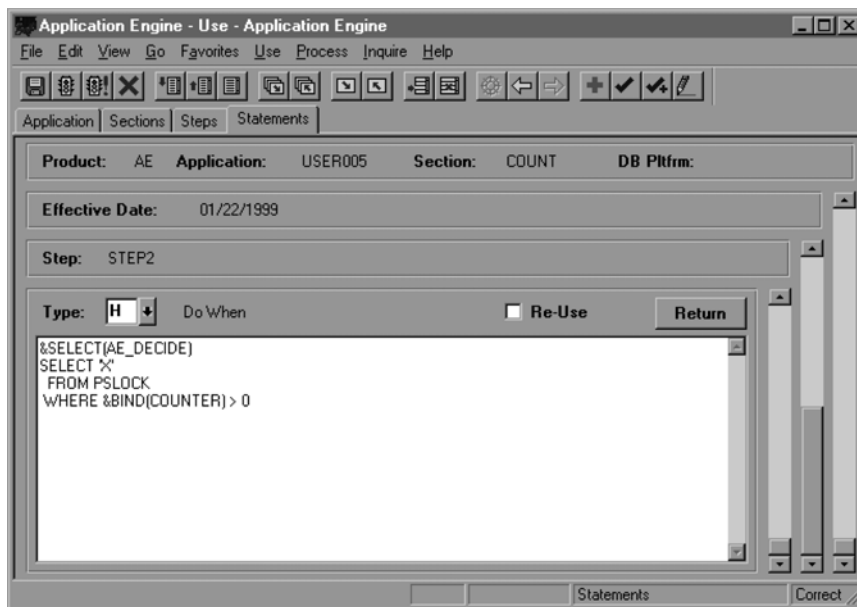


Figure 40.24 DO When statement text

40.1.3 PSLOCK and decision making

We are using a PeopleSoft-delivered table called PSLOCK which consists of one row. We're going to use the PSLOCK table as a placeholder instead of actually selecting data from the table itself. If you are an Oracle user, you may be familiar with this technique against the DUAL table. Although the main function of the PSLOCK table

has little to do with Application Engine, it is ideal for decision-making functions such as this. You will find this frequently in PeopleSoft A/E processes. Let's take a closer look at our statement text:

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

First, let's look at the SQL `Select` statement. If the cache field `COUNTER` has a value greater than zero, a single row with the character 'X' will be returned. This value will then be assigned to the cache field `AE_DECIDE` using the `&SELECT` function. If the `COUNTER` value is not greater than zero, the `AE_DECIDE` cache field will have a default value of blank, and no rows will be returned by the `Select` statement.

If a `DO When Select` statement returns rows, the `DO` section will be performed. This is a simple but effective decision-making tool. Now, we need to link our `MSG` Section to this step.

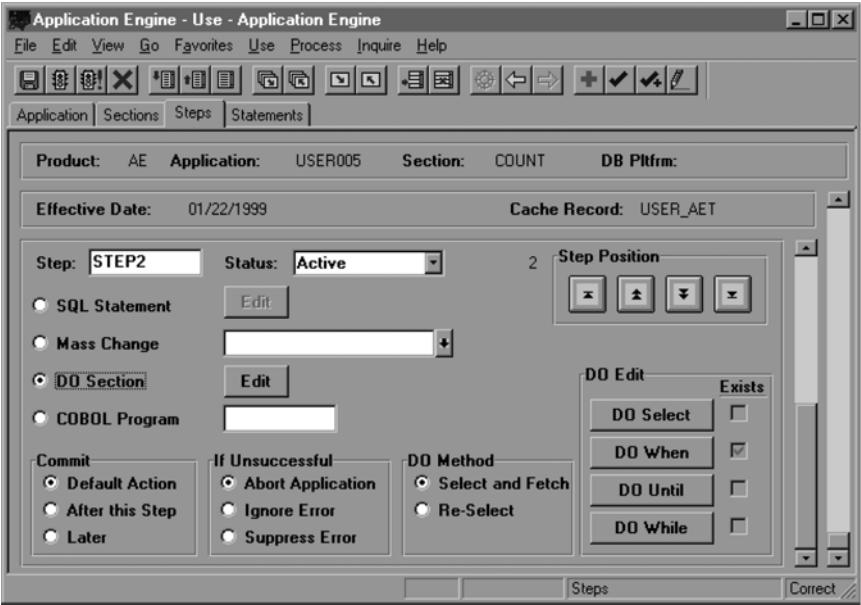


Figure 40.25 Setting the DO section for our DO When statement type

Click on the `DO` section edit box.. You may have noticed the `DO` Edit group box on the lower right side of the panel. Because we're using the `DO When` statement type, the `Exists` checkbox is automatically filled in next to the `DO When` push button. Press-

ing the DO When push button has the same affect as clicking on the Statements folder tab. It's simply an alternate method of navigation.

Set the product and application and then click on the section drop-down box (figure 40.26).

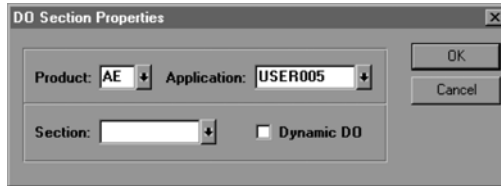


Figure 40.26
DO section properties dialog box

Highlight and click on the MSG section (figure 40.27).

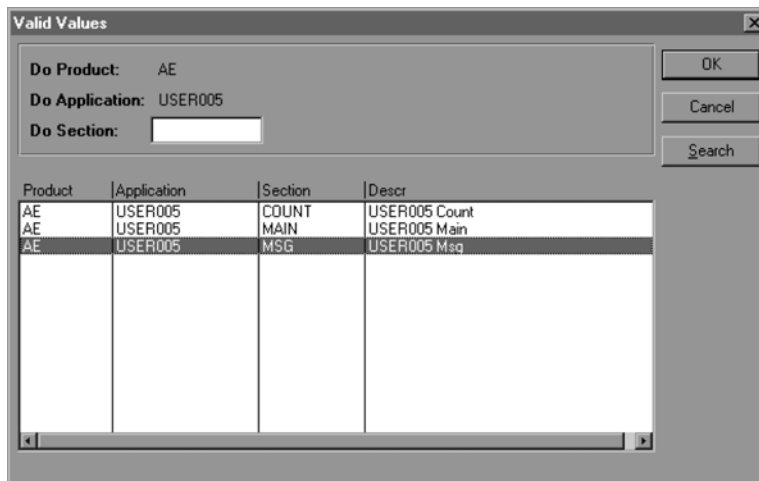


Figure 40.27 Selecting the DO section

Once selected, we return to the DO Section Properties box with our section MSG filled in (figure 40.28). Click the OK button.

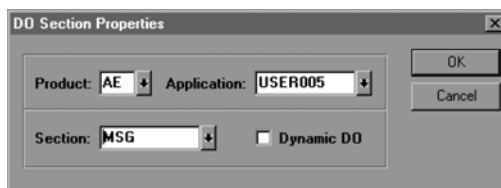


Figure 40.28
The completed DO Section
Properties dialog box

When we return to the Step Definition screen, we see the MSG section displayed to the right of the DO section radio button. For each row returned by the STEP2 DO When statement, the MSG section will be executed. This means only tables with a row count greater than zero will be displayed in the message log. We're ready to test our program.

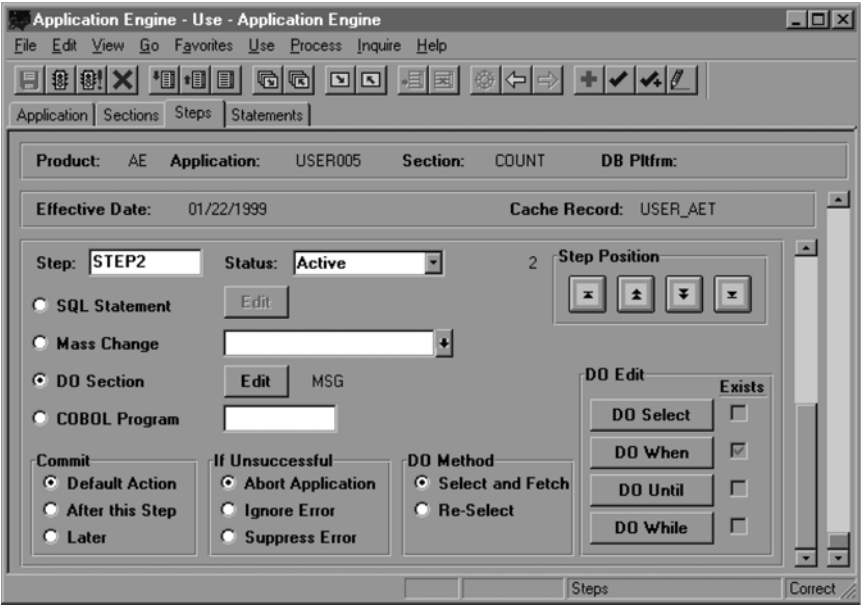


Figure 40.29 The DO section has been completed

Return to the Process Request panel and add the Run Control ID #USER005. Click the OK button.

Navigation: Go →PeopleTools →Application Engine →Process →Request →Request →Add



Figure 40.30
Adding the Run Control ID

When the Process Request panel appears, click on the Field edit box and enter the cache field FIELDNAME. Now, enter the value PAY_END_DT (figure 40.31). Save the record and click on the Traffic Signal to initiate the process request.

The screenshot shows the 'Application Engine - Process - Request' window. The 'Request' tab is active. The 'Operator Id' is 'PS' and the 'Run Control ID' is '#USER005'. The 'Request Number' is '1'. The 'Process Frequency' is set to 'Always'. The 'Product' is 'AE' and the 'Application' is 'USER005'. Below these fields is a 'Fields' section with a table showing the field 'FIELDNAME' and its value 'PAY_END_DT'. The 'Update' button is visible at the bottom right.

Field	Value
FIELDNAME	PAY_END_DT

Figure 40.31 Assigning an initial value to the cache field

Once again, highlight the AEADHOC process (figure 40.32) and click OK.