PeopleSoft®

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Engine

June 2004

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Engine
SKU PT845APE-B 0604

Open Source Disclosure

PeopleSoft takes no responsibility for its use or distribution of any open source or shareware software or documentation and disclaims any and all liability or damages resulting from use of said software or documentation. The following open source software may be used in PeopleSoft products and the following disclaimers are provided.

Apache Software Foundation

This product includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright (c) 1999-2000 The Apache Software Foundation. All rights reserved.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

OpenSSL

Copyright (c) 1998-2003 The OpenSSL Project. All rights reserved.

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SSLeay

Copyright (c) 1995-1998 Eric Young. All rights reserved.

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Loki Library

Copyright (c) 2001 by Andrei Alexandrescu. This code accompanies the book:

Alexandrescu, Andrei. "Modern C++ Design: Generic Programming and Design Patterns Applied". Copyright (c) 2001. Addison-Wesley. Permission to use, copy, modify, distribute and sell this software for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

Contents

About This PeopleBook

PeopleBooks provide you with the information that you need to implement and use PeopleSoft applications.

This preface discusses:

- PeopleSoft application prerequisites.
- PeopleSoft application fundamentals.
- Related documentation.
- Typographical conventions and visual cues.
- Comments and suggestions.
- Common elements in PeopleBooks.

Note. PeopleBooks document only page elements that require additional explanation. If a page element is not documented with the process or task in which it is used, then either it requires no additional explanation or it is documented with common elements for the section, chapter, PeopleBook, or product line. Elements that are common to all PeopleSoft applications are defined in this preface.

PeopleSoft Application Prerequisites

To benefit fully from the information that is covered in these books, you should have a basic understanding of how to use PeopleSoft applications.

See *Enterprise PeopleTools 8.45 PeopleBook: Using PeopleSoft Applications*.

You might also want to complete at least one PeopleSoft introductory training course.

You should be familiar with navigating the system and adding, updating, and deleting information by using PeopleSoft windows, menus, and pages. You should also be comfortable using the World Wide Web and the Microsoft Windows or Windows NT graphical user interface.

These books do not review navigation and other basics. They present the information that you need to use the system and implement your PeopleSoft applications most effectively.

PeopleSoft Application Fundamentals

Each application PeopleBook provides implementation and processing information for your PeopleSoft database. However, additional, essential information describing the setup and design of your system appears in a companion volume of documentation called the application fundamentals PeopleBook. Each PeopleSoft product line has its own version of this documentation.

The application fundamentals PeopleBook consists of important topics that apply to many or all PeopleSoft applications across a product line. Whether you are implementing a single application, some combination of applications within the product line, or the entire product line, you should be familiar with the contents of this central PeopleBook. It is the starting point for fundamentals, such as setting up control tables and administering security.

Related Documentation

This section discusses how to:

- Obtain documentation updates.
- Order printed documentation.

Obtaining Documentation Updates

You can find updates and additional documentation for this release, as well as previous releases, on the PeopleSoft Customer Connection website. Through the Documentation section of PeopleSoft Customer Connection, you can download files to add to your PeopleBook Library. You'll find a variety of useful and timely materials, including updates to the full PeopleSoft documentation that is delivered on your PeopleBooks CD-ROM.

Important! Before you upgrade, you must check PeopleSoft Customer Connection for updates to the upgrade instructions. PeopleSoft continually posts updates as the upgrade process is refined.

See Also

PeopleSoft Customer Connection, https://www.peoplesoft.com/corp/en/login.jsp

Ordering Printed Documentation

You can order printed, bound volumes of the complete PeopleSoft documentation that is delivered on your PeopleBooks CD-ROM. PeopleSoft makes printed documentation available for each major release shortly after the software is shipped. Customers and partners can order printed PeopleSoft documentation by using any of these methods:

- Web
- Telephone
- Email

Web

From the Documentation section of the PeopleSoft Customer Connection website, access the PeopleBooks Press website under the Ordering PeopleBooks topic. The PeopleBooks Press website is a joint venture between PeopleSoft and MMA Partners, the book print vendor. Use a credit card, money order, cashier's check, or purchase order to place your order.

Telephone

Contact MMA Partners at 877 588 2525.

Email

Send email to MMA Partners at peoplesoftpress@mmapartner.com.

See Also

PeopleSoft Customer Connection, https://www.peoplesoft.com/corp/en/login.jsp

Typographical Conventions and Visual Cues

This section discusses:

- Typographical conventions.
- Visual cues.
- Country, region, and industry identifiers.
- Currency codes.

Typographical Conventions

This table contains the typographical conventions that are used in PeopleBooks:

Typographical Convention or Visual Cue	Description
Bold	Indicates PeopleCode function names, method names, language constructs, and PeopleCode reserved words that must be included literally in the function call.
Italics	Indicates field values, emphasis, and PeopleSoft or other book-length publication titles. In PeopleCode syntax, italic items are placeholders for arguments that your program must supply. We also use italics when we refer to words as words or letters as letters, as in the following: Enter the letter *O*.
KEY+KEY	Indicates a key combination action. For example, a plus sign (+) between keys means that you must hold down the first key while you press the second key. For ALT+W, hold down the ALT key while you press the W key.
`Monospace font`	Indicates a PeopleCode program or other code example.
" " (quotation marks)	Indicate chapter titles in cross-references and words that are used differently from their intended meanings.
. . . (ellipses)	Indicate that the preceding item or series can be repeated any number of times in PeopleCode syntax.
{ } (curly braces)	Indicate a choice between two options in PeopleCode syntax. Options are separated by a pipe (|).

Typographical Convention or Visual Cue	Description
[ ] (square brackets)	Indicate optional items in PeopleCode syntax.
& (ampersand)	When placed before a parameter in PeopleCode syntax, an ampersand indicates that the parameter is an already instantiated object. Ampersands also precede all PeopleCode variables.

Visual Cues

PeopleBooks contain the following visual cues.

Notes

Notes indicate information that you should pay particular attention to as you work with the PeopleSoft system.

Note. Example of a note.

If the note is preceded by *Important!*, the note is crucial and includes information that concerns what you must do for the system to function properly.

Important! Example of an important note.

Warnings

Warnings indicate crucial configuration considerations. Pay close attention to warning messages.

Warning! Example of a warning.

Cross-References

PeopleBooks provide cross-references either under the heading “See Also” or on a separate line preceded by the word *See*. Cross-references lead to other documentation that is pertinent to the immediately preceding documentation.

Country, Region, and Industry Identifiers

Information that applies only to a specific country, region, or industry is preceded by a standard identifier in parentheses. This identifier typically appears at the beginning of a section heading, but it may also appear at the beginning of a note or other text.

Example of a country-specific heading: “(FRA) Hiring an Employee”

Example of a region-specific heading: “(Latin America) Setting Up Depreciation”

Country Identifiers

Countries are identified with the International Organization for Standardization (ISO) country code.

See *About These PeopleBooks*, “ISO Country and Currency Codes,” ISO Country Codes.

Region Identifiers

Regions are identified by the region name. The following region identifiers may appear in PeopleBooks:

- Asia Pacific
- Europe
- Latin America
- North America

Industry Identifiers

Industries are identified by the industry name or by an abbreviation for that industry. The following industry identifiers may appear in PeopleBooks:

- USF (U.S. Federal)
- E&G (Education and Government)

Currency Codes

Monetary amounts are identified by the ISO currency code.

Appendix A, "ISO Country and Currency Codes" ISO Currency Codes.

Comments and Suggestions

Your comments are important to us. We encourage you to tell us what you like, or what you would like to see changed about PeopleBooks and other PeopleSoft reference and training materials. Please send your suggestions to:

PeopleSoft Product Documentation Manager PeopleSoft, Inc. 4460 Hacienda Drive Pleasanton, CA 94588

Or send email comments to doc@peoplesoft.com.

While we cannot guarantee to answer every email message, we will pay careful attention to your comments and suggestions.

Common Elements in These PeopleBooks

As of Date	The last date for which a report or process includes data.
Business Unit	An ID that represents a high-level organization of business information. You can use a business unit to define regional or departmental units within a larger organization.
Description	Enter up to 30 characters of text.
Effective Date	The date on which a table row becomes effective; the date that an action begins. For example, to close out a ledger on June 30, the effective date for the ledger closing would be July 1. This date also determines when

	you can view and change the information. Pages or panels and batch processes that use the information use the current row.
Once, Always, and **Don't Run**	Select Once to run the request the next time the batch process runs. After the batch process runs, the process frequency is automatically set to Don't Run. Select Always to run the request every time the batch process runs. Select Don't Run to ignore the request when the batch process runs.
Report Manager	Click to access the Report List page, where you can view report content, check the status of a report, and see content detail messages (which show you a description of the report and the distribution list).
Process Monitor	Click to access the Process List page, where you can view the status of submitted process requests.
Run	Click to access the Process Scheduler request page, where you can specify the location where a process or job runs and the process output format.
Request ID	An ID that represents a set of selection criteria for a report or process.
User ID	An ID that represents the person who generates a transaction.
SetID	An ID that represents a set of control table information, or TableSets. TableSets enable you to share control table information and processing options among business units. The goal is to minimize redundant data and system maintenance tasks. When you assign a setID to a record group in a business unit, you indicate that all of the tables in the record group are shared between that business unit and any other business unit that also assigns that setID to that record group. For example, you can define a group of common job codes that are shared between several business units. Each business unit that shares the job codes is assigned the same setID for that record group.
Short Description	Enter up to 15 characters of text.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Process Scheduler

Enterprise PeopleTools 8.45 PeopleBook: Using PeopleSoft Applications

PeopleSoft Application Engine Preface

This PeopleBook describes PeopleSoft Application Engine.

PeopleSoft Application Engine

PeopleSoft Application Engine is designed to help you develop, test, and run background SQL processing programs. This PeopleBook explains the concepts and advantages of PeopleSoft Application Engine, how to develop Application Engine programs in PeopleSoft Application Designer, how to run and debug programs, and the use of the special tools to maintain your programs.

The “About These PeopleBooks” preface contains general product line information, such as related documentation, common page elements, and typographical conventions.

Note. *DB2 UDB for OS/390 and z/OS* is the official IBM name for the DBMS. In the current PeopleTools release, PeopleSoft no longer supports the OS/390 operating system, only z/OS, its replacement. For the sake of brevity, this PeopleBook sometimes refers to DB2 UDB for OS/390 and z/OS as *DB2 z/OS*, and it sometimes refers to DB2 UDB for Linux, UNIX, and Windows as *DB2 UNIX/NT*.

CHAPTER 1

Getting Started With PeopleSoft Application Engine

This chapter provides an overview of PeopleSoft Application Engine and discusses:

- PeopleSoft Application Engine implementation
- Other sources of information

Application Engine Overview

PeopleSoft Application Engine is a PeopleTool designed to help you develop background SQL processing programs. This tool is intended to be used by developers with knowledge of SQL, SQL tools and PeopleTools.

PeopleSoft Application Engine offers you an alternative to writing COBOL, or SQR programs for background SQL processing. While PeopleSoft Application Engine does not generate, parse, or understand SQL, it does execute SQL that you provide.

PeopleSoft Application Engine Implementation

This section provides information to consider before you begin to use PeopleSoft Application Engine.

Implementation of PeopleSoft Application Engine can be divided into the following activities:

- Set up properties.
- Specify actions.
- Create temporary table instances.
- Set up debug options.
- Enable application engine tracing.

Setting Up Properties

To set up PeopleSoft Application Engine properties, you perform the following steps:

Step	Reference
1. Set up program properties.	See Chapter 3, “Creating Application Engine Programs,” Setting Program Properties, page 21.
2. Set up section properties.	See Chapter 3, “Creating Application Engine Programs,” Setting Section Properties, page 27.
3. Set up step properties.	See Chapter 3, “Creating Application Engine Programs,” Setting Step Properties, page 28.
4. Set up action properties.	See Chapter 3, “Creating Application Engine Programs,” Setting Action Properties, page 31.

Specifying Actions

To modify the action properties, you perform the following steps:

Step	Reference
1. Specify SQL actions.	See Chapter 3, “Creating Application Engine Programs,” Specifying SQL Actions, page 32.
2. Specify Do actions.	See Chapter 3, “Creating Application Engine Programs,” Specifying Do Actions, page 33.
3. Specify PeopleCode actions.	See Chapter 3, “Creating Application Engine Programs,” Specifying PeopleCode Actions, page 35.
4. Specify Call Section actions.	See Chapter 3, “Creating Application Engine Programs,” Specifying Call Section Actions, page 36.
5. Specify Log Message actions.	See Chapter 3, “Creating Application Engine Programs,” Specifying Log Message Actions, page 37.

Creating Temporary Table Instances

To set up temporary tables to improve performance, you perform the following steps:

Step	Reference
1. Define temporary tables.	See Chapter 9, “Using Temporary Tables,” Defining Temporary Tables, page 138.
2. Set up the number of temporary table instances.	See Chapter 9, “Using Temporary Tables,” Setting the Number of Temporary Table Instances, page 138.
3. Build table instances.	See Chapter 9, “Using Temporary Tables,” Building Table Instances, page 140.

Setting Up Debugging Options

To set up debugging options for Application Engine programs, you perform the following steps:

Step	Reference
1. Enable the PeopleSoft Application Engine debugger.	See Chapter 6, "Managing Application Engine Programs," Enabling the PeopleSoft Application Engine Debugger, page 105.
2. Set up debugging options.	See Chapter 6, "Managing Application Engine Programs," Setting Debugging Options, page 106.

Enabling Application Engine Tracing

To trace Application Engine programs, you perform the following steps:

Step	Reference
1. Set command line options.	See Chapter 8, "Tracing Application Engine Programs," Setting Command Line Options, page 131.
2. Set parameters in server configuration files.	See Chapter 8, "Tracing Application Engine Programs," Setting Parameters in Server Configuration Files, page 132.
3. Set options in Configuration Manager	See Chapter 8, "Tracing Application Engine Programs," Setting Options in PeopleSoft Configuration Manager, page 132.

Other Sources of Information

In addition to implementation considerations presented in this chapter, take advantage of all PeopleSoft sources of information, including the installation guides, release notes, and PeopleBooks.

See Also

"PeopleSoft Application Engine Preface," page xvii

Enterprise PeopleTools 8.45 PeopleBook: Getting Started with Enterprise PeopleTools

CHAPTER 2

Understanding PeopleSoft Application Engine

You use PeopleSoft Application Engine to develop batch or online programs that perform high-volume, background processing against your data.

This chapter discusses:

- PeopleSoft Application Engine fundamentals.
- Meta-Structured Query Language (SQL).
- Application Engine program elements.
- Application Engine program types.

PeopleSoft Application Engine Fundamentals

PeopleSoft Application Engine comprises two distinct components—a designer where you define your batch program and the runtime environment where you run and monitor your program.

In PeopleSoft Application Engine, a *program* is a set of SQL statements, PeopleCode, and program control actions that enable looping and conditional logic. A program is defined in PeopleSoft Application Designer and performs a business process. You can use PeopleSoft Application Engine for straight, row-by-row processing, but the most efficient Application Engine programs are written to perform set-based processing.

PeopleSoft Application Engine does not generate SQL or PeopleCode. It executes the SQL and PeopleCode that you include in an Application Engine action as part of your program.

PeopleSoft Application Engine is designed for batch processing where you have data that must be processed without user intervention—for example, calculating salaries in payroll processing (although not printing the checks). Another example might be converting money from one currency to another.

Meta-SQL

You can write SQL within PeopleSoft Application Engine, or you can copy SQL statements into Application Engine from any SQL utility with few, if any, changes. This enables you to write and tune SQL statements before you try to incorporate them into an Application Engine program.

Database platforms can have different syntax rules, especially in regard to date, time, and other numeric calculations. Generally, you can work around syntax differences using PeopleSoft meta-SQL, which PeopleSoft Application Engine supports. Meta-SQL is a set of predefined terms (meta-strings), designed to replace relational database management system (RDBMS)-specific SQL syntax with a common syntax.

In addition, PeopleSoft meta-SQL enables you to dynamically generate portions of SQL code. For example, to join two tables based on their common keys, use the following meta-string:

```
%Join(COMMON_KEYS, PSAESECTDEFN ABC,  PSAESTEPDEFN XYZ )
```

At runtime, the function would be expanded into the following:

```
ABC.AE_APPLID = XYZ.AE_APPLID
AND ABC.AE_SECTION = XYZ.AE_SECTION
AND ABC.DBTYPE = XYZ.DBTYPE
AND ABC.EFFDT = XYZ.EFFDT
```

Application Engine Program Elements

A PeopleSoft Application Engine program comprises the set of processes to execute a given task, and is made up of several key elements:

- Sections.
- Steps.
- Actions.
- State records.

Sections

Sections comprise one or more steps and are equivalent to a COBOL paragraph or a Structured Query Report (SQR) procedure. All Application Engine programs must contain at least one section entitled *MAIN*.

A section is a set of ordered steps that gets executed as part of a program. You can call sections (and other programs) from steps within other sections.

A program must contain at least one section. The execution of the program always starts with the section defined as MAIN.

Steps

Steps are the smallest unit of work that can be committed within a program. Although you can use a step to execute a PeopleCode command or log a message, typically you use a step to execute a SQL statement or to call another section. The SQL or PeopleCode that a step executes are the actions within the step.

When a section gets called, its steps execute sequentially. Every program begins by executing the first step of the required section called MAIN and ends after the last step in the last section completes successfully.

Actions

There are multiple types of actions that you can specify to include within a step. It is common to have multiple actions associated with a single step.

Do Actions

Do actions contain a SQL **Select** statement designed to return results on which subsequent actions depend. For instance, if a **Select** statement returns no rows, subsequent actions may not need to execute. A Do action is equivalent to a COBOL **Perform** statement and has similar constructs.

The four types of Do actions are:

- Do While
- Do When
- Do Select
- Do Until

SQL

Most SQL actions contain a single SQL statement. These actions can perform the following types of SQL statements:

- **Update**
- **Delete**
- **Insert**
- **Select**

  The SQL action differs from the Do actions, which also contain SQL, in that the SQL action does not control the flow of the program.

PeopleCode

You can include PeopleCode in the PeopleCode action. PeopleSoft Application Engine PeopleCode provides an excellent way to build dynamic SQL, perform simple if/else edits, set defaults, and other operations that don't require a trip to the database. It also enables you to reference and change active Application Engine state records.

Most importantly, PeopleCode provides access to the PeopleSoft integration technologies such as PeopleSoft Integration Broker, Component Interfaces, Business Interlinks, and file processing.

Log Message

You use a Log Message action to write a message to the message log based on a condition in your program. This gives your program multilanguage capability. The system stores the message generically as a message set, message number, and parameter values. When a user views the messages using the Application Engine Message Log page, the system retrieves the appropriate message string from the message catalog based on the user's language preference.

Call Section

You can also insert an action that calls another section. The called section can be in the same program as the calling section, or it can be in an external program. This enables you to chunk your program into more maintainable, reusable pieces. If a section already exists in one program, rather than copying it into another program, just call it.

Note. PeopleSoft Application Engine supports up to 99 levels of nested Call Section actions. For example, the first called section can call a second, which can call a third, and so on, up to 99 calls.

State Records

A state record is a PeopleSoft record that must be created and maintained by the Application Engine developer. This record defines the fields a program uses to pass values from one action to another. Think of the fields of the Application Engine state record as the working storage for your Application Engine program.

An Application Engine state record can be either a physical record or a work record, and any number of state records can be associated with a program. Physical state records must be keyed by process instance.

Application Engine Program Types

This section discusses:

- Application Engine program types.
- Daemon program type.
- Transform program type.

Application Engine Program Types

There are five types of Application Engine programs. You specify the type in the Program Properties dialog box for your program definition. The types are:

- Standard, which is a normal entry-point program.
- Upgrade Only, which is used in PeopleSoft upgrade utilities.
- Import Only, which is used by PeopleSoft import utilities.
- Daemon Only, a type of program used as a daemon process.
- Transform Only, a program type used to support Extensible Stylesheet Language Transformations (XSLT).

Daemon Program Type

PeopleSoft Application Engine provides a daemon process, called PSDAEMON, that runs continuously when PeopleSoft Process Scheduler is running, and is intended for recurring jobs. It polls the system, checking for certain conditions to occur. A predefined set of conditions is an *event*. When the conditions are true, PSDAEMON schedules a process to handle the event.

PSDAEMON supports limited tracing, because it runs indefinitely. Specifically, it only allows PeopleSoft Application Engine tracing at the step and SQL levels, in addition to the standard PeopleSoft SQL and PeopleCode tracing. Other options, such as Timings and DB Optimizer tracing, are not supported.

You activate PSDAEMON in PeopleSoft Process Scheduler or from the command line.

Note. One PSDAEMON process can run for each row in the PS_SERVERDEFN table. The PS_SERVERDEFN.DAEMONENABLED field must be set to 1.

Starting PSDAEMON from the Command Line

The command line syntax is:

psdaemon [-CT *database_type*] [-CD *database_name*] [-CO *userID*] [-CP *password*] -R *server_name*

Use the -R option to query PS_SERVERDEFN, obtaining the daemon group, sleep time, and recycle count (terminate after *N* iterations). *Server_name* is the key value for PS_SERVERDEFN. You do not need to pass ProcessInstance (-I) or AE Program ID (-AI).

Starting a Daemon Program from PeopleSoft Process Scheduler

Before starting a daemon Application Engine program, you must add the program to the Daemon Group page in PeopleSoft Process Scheduler.

To add a daemon program:

1. Select PeopleTools, Process Scheduler, Daemon Group.
2. Select the Add New Value page.
3. Enter a daemon procedure group name, and click Add.
4. On the Daemon Group page, add the appropriate programs to the program name list.

Restarting the AEDAEMONMGR Program

AEDAEMONMGR is a restartable Application Engine program, which commits after each daemon procedure. When the PSDAEMON executes, it determines whether it must restart AEDAEMONMGR following an abnormal end to a program.

If a restart is not required, PSDAEMON assigns a new process instance and runs AEDAEMONMGR from the beginning. Because of this design, PeopleSoft Process Scheduler does not have to determine whether PSDAEMON exited due to an error or because it had reached the recycle count.

AEDAEMONMGR uses the Daemon Group page value to get related daemon procedures from PS_DAEMONGROUP in order, and then it initiates each procedure. After all procedures have been executed, AEDAEMONMGR logs a sleep message and returns control to PSDAEMON. The sleep time is used only to log an informational message at the end of each cycle, for example "Sleeping for N minutes...". A message is also logged at the beginning of each cycle, so an administrator can monitor the runtime and sleep-time of a specific PSDAEMON process.

If an error occurs in AEDAEMONMGR; if the recycle count has been reached; or if PSSERVERSTAT.DAEMONACTION = ’1’ (indicating that PeopleSoft Process Scheduler is idle), PSDAEMON exits. Otherwise, it sleeps for the requested number of minutes, then calls AEDAEMONMGR again.

Using PSDAEMON to Start Parallel Processing

Within a daemon group, programs are invoked sequentially, and one program does not execute until the previous program has completed. The programs contained in a daemon group should be quick programs that scan information to find events. When an event is discovered, the daemon program can use the ProcessRequest class to invoke programs that are not of the daemon type. These *non-daemon type* Application Engine programs can execute in parallel. For that reason, do not include application-specific processing in a PSDAEMON type program.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Process Scheduler, “Defining PeopleSoft Process Scheduler Support Information,” Defining Process Type Definitions

Transform Program Type

Transform Only type programs enable different systems to communicate with one another by transforming messages into appropriate formats. When you specify an Application Engine program as a Transform Only program, you must specify actions of type XSLT or PeopleCode. You can use transform programs to do any of the following:

- Apply a transformation to a message to make its structure comply with the target system's requirements.
- Perform a data translation on a message so its data is represented according to the target system's conventions.
- Determine whether to pass a message through to its target, by filtering it based on its content.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker, "Applying Filtering, Transformation and Translation," Developing Transform Programs

CHAPTER 3

Creating Application Engine Programs

An Application Engine program includes a logically ordered set of sections, steps, and actions. An executable program must contain at least one section, called MAIN, used to identify the starting point of the program; it should contain at least one step; each step should contain at least one action.

This chapter discusses how to:

- View Application Engine programs.
- Filter view contents.
- Print program and flow definitions.
- Create, open, and rename programs.
- Copy or move program elements.
- Test Application Engine programs.
- Set program properties.
- Add sections.
- Add steps.
- Specify actions.

Viewing Application Engine Programs

This section discusses how to:

- Use Definition view.
- Use Program Flow view.
- Switch between Definition and Program Flow views.
- Use the Refresh option.

Using Definition View

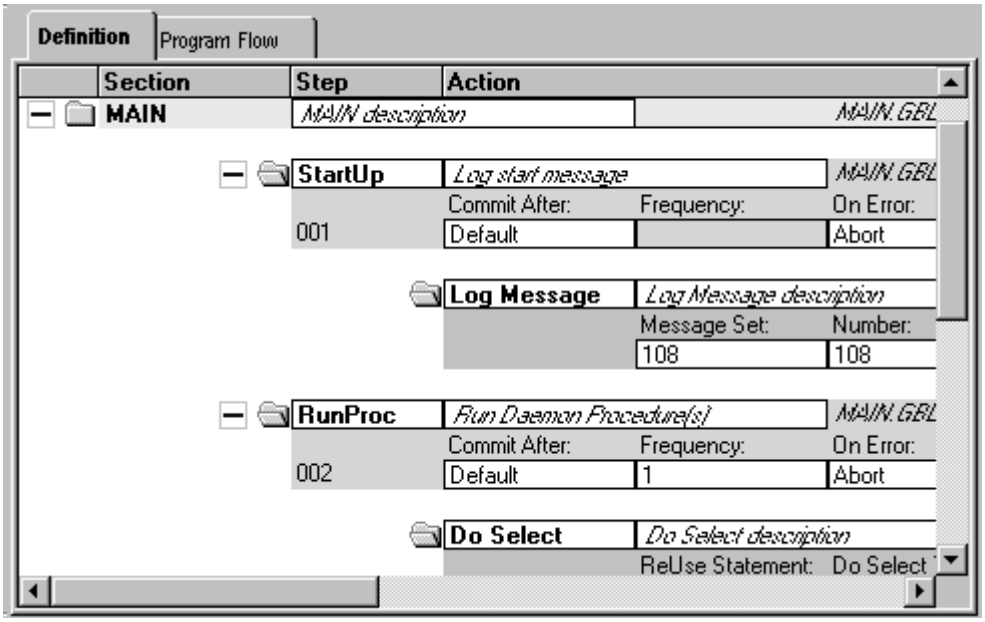
You use Definition view to create definitions within a defined hierarchical structure, in which nodes represent the definitions. A node is the visual representation of a section, step, or action that you can select, collapse, modify, and so on.

The sections that appear in Definition view do not necessarily appear in the order that they execute. To see the actual order in which the sections execute, switch to Program Flow view.

Besides using the mouse, you can navigate in this view using the following keyboard combinations:

- Press CTRL+HOME to scroll to the top of the program definition and select the first node.
- Press CTRL+END to scroll to the end of the program definition and select the last visible node.
- Press TAB to move from the currently selected field to the next updateable field.
- Press CTRL+DOWN ARROW to move from the currently selected node to the next node.
- Press CTRL+UP ARROW to move from the currently selected node to the previous node.

The following illustration shows Definition view:



PeopleSoft Application Designer Definition view

Definition View Pop-up Menu

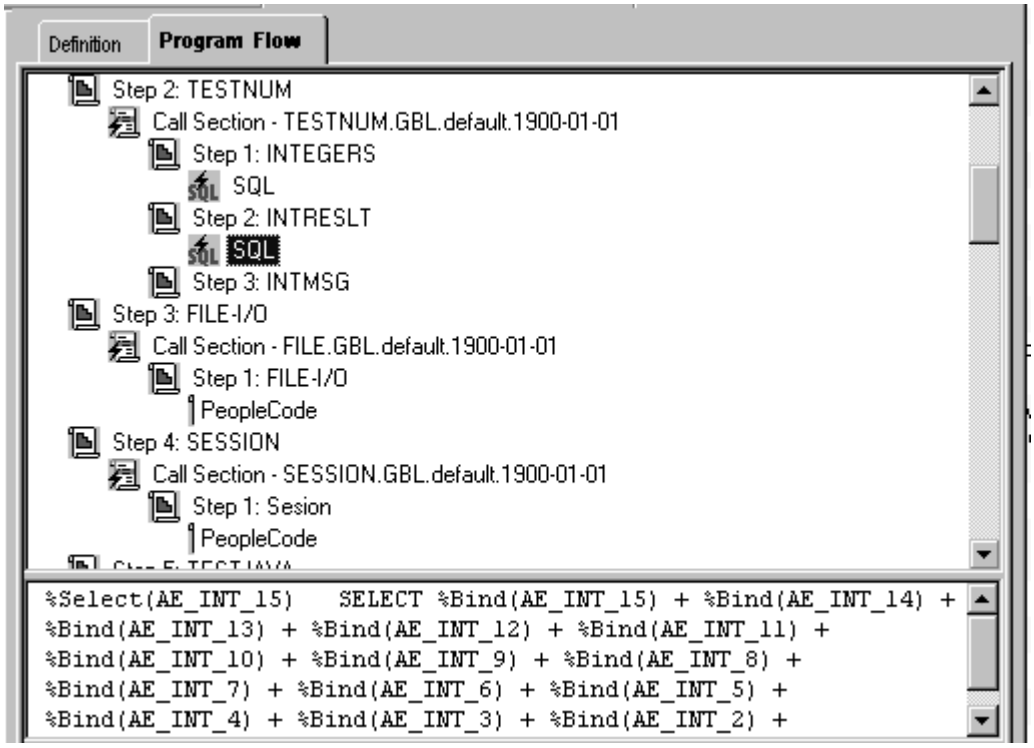
The following table describes each item you see when you right-click a Definition view window. Certain menu items are enabled only when a particular definition is selected.

Menu Command	Description
View PeopleCode	Launches the PeopleCode Editor with the appropriate PeopleCode loaded. Enabled when a PeopleCode action is selected.
View SQL	Launches the SQL Editor with the appropriate SQL loaded. Enabled when an action containing SQL is selected.
View XSLT	Launches the SQL Editor with the related Extensible Stylesheet Language Transformations (XSLT) text loaded. Enabled for Transform Only program types only, when an XSLT action is selected.
Cut	Removes the selected item and copies it to a clipboard. Here, the word *clipboard* refers to a PeopleTools-only repository for sharing PeopleTools objects. You cannot copy or paste into another program.
Copy	Copies the selected item.

Menu Command	Description
Paste	Pastes the contents of the PeopleTools clipboard (the most recently cut or copied item) to the current location of the cursor.
Delete	Removes the currently selected node from the program definition.
Refresh View	Refreshes the current view and reorders the definition objects as necessary.
Show Comment	Reveals the comments associated with the selected definition object.
Insert Section	Inserts a new section into the current program, at the place where the cursor is positioned. This option is enabled only when you have MAIN or another section selected.
Insert Step/Action	Inserts a new step and action within the currently selected section. This option is enabled only when you have a section or a step selected.
Insert Action	Inserts a new action within the currently selected step. This option is enabled only when you have a step or action selected.
Jump to This Program Flow	Switches to the Program Flow view with the first occurrence of the currently selected definition in focus.
Print	Displays the print dialog box for the definition view.
Insert Section Into Project	Applies to sections. Inserts the currently selected section into the current project.

Using Program Flow View

Program Flow view is a read-only view that shows the expected sequence of steps to be executed at runtime for the program you are developing. The following illustration shows Program Flow view:



PeopleSoft Application Designer Program Flow view

You can control the amount of detail that appears for each definition by clicking it to expand to the next level. You can also view the SQL or PeopleCode in the lower (splitter) window area by clicking on the lower window.

To display the pop-up menu for a node, right-click the node. You do not have to select the node first.

You can also display the comments associated with definitions by selecting View, Show All Comments, or for a particular node, right-click and select Show Comment.

You can double-click SQL or PeopleCode statements to launch the editors.

Program Flow Pop-up Menu

The following table contains each pop-up menu item in Program Flow view.

Menu Command	Description
View PeopleCode	Launches the PeopleCode Editor with the appropriate PeopleCode loaded. Enabled when a PeopleCode action is selected.
View SQL	Launches the SQL Editor with the appropriate SQL loaded. Enabled when an action containing SQL is selected.
Refresh View	Refreshes the current view and reorders the definition objects as necessary.
Show Comment	Reveals the comments for a single definition object that appears in the Program Flow view.

Menu Command	Description
Jump to This Definition	Switches to the Definition view with the first occurrence of the currently selected definition object in focus.
Print	Launches the print dialog box for the program view.

Switching Between Definition and Program Flow Views

By default, navigation within either view does not affect the currently active row in the other view. This enables you to retain your place in one view while scrolling around in the other.

To switch between the two views, you can use any of the following methods.

- View tabs.

  As with any tabbed interface, if you select a tab, the associated view interface becomes active. When you return to the previous view, it remains positioned on the current or last selected node within the program when you switched. This is true whether you selected the item or just placed the cursor within an edit box.

- View menu.

  Select a section or step in the current view (note that selecting an action does not enable this functionality—you can only jump from parent nodes). Then select View, Jump to Program Flow or View, Jump to Definition, depending on the view that is currently active. When you select one of these commands, the focus of the target view depends on what you have selected in the previous view. For example, if you have section C, step 4 selected in Definition view, and you select View, Jump to Program Flow, section C, step 4 is the focus of the Program Flow window. If the selected item is in a program that is not already open, PeopleSoft Application Engine opens the appropriate program, then navigates to the requested node in the view window.

- Pop-up menu.

  The same commands as the View menu are also available from the pop-up menu.

Switching Within Program Flow View

While you are in Program Flow view, you can select these options from the pop-up menu:

- Go to Next Reference

  Select to switch to the next reference of a particular definition object. This helps you to quickly navigate through a program. For instance, if references to section C, step 4 appear three times because there are multiple calls to this object at runtime, you select Go to Next Reference to quickly and easily navigate to each reference, or call.

- Jump to this Definition

  Select to go directly to the definition node in Definition view that pertains to the current selection in the Program Flow view.

Using the Refresh Option

As you develop an Application Engine program, you may be inserting, renaming, and deleting definitions. In a large program, it can be easy to lose your place or become disoriented. The Refresh option reorders all the nodes for the current definition according to the following logic:

- For standard program definitions, the MAIN section is always displayed first (Library program types do not contain a MAIN section because they contain only callable sections).

The remaining sections appear alphabetically by name, which makes it easier to locate a section within the program definition. The system, at runtime, executes sections through Call Section actions within steps, not by the order in which sections are defined.

- Steps are never automatically reordered in Definition view, and, at runtime, they execute in the sequence in which you define them.
- Actions are always logically reordered within a step, based on their action type, which defines their runtime sequence.

Note. When you save a modified definition, the system automatically refreshes the view.

PeopleSoft Application Engine inserts any delete requests for a given section into the current project, regardless of the Tools, Options setting in PeopleSoft Application Designer.

For example, suppose you delete a section node from the current Application Engine program, and then you reinsert a section node and rename it to the same name as the section you just deleted. The section object is not inserted into the project regardless of your Tools, Options setting. This is because a delete action already exists for this object. To resolve this situation, either manually remove the delete request before inserting the new copy request or manually reset the proper flags in the upgrade project that changes the action type from delete to copy.

Filtering View Contents

Section filtering options enable you to filter the current view so that you see only sections and steps based on specified criteria.

To enable or modify the filtering options, select View, Section Filtering. You can select from the following filtering options:

Menu Command	Description
No Filtering	Select to see all objects in your program regardless of any section attributes, such as Market, Database Type, Effective Date, and Effective Status.
Default	Select to display the definition filter according to the default filtering criteria. If you change the value of any filter option and click OK, you have defined a custom filter.
Custom	Select to display the definition filter dialog box and define custom filtering options for the current view.

Behavior of Section Filtering Options

When using the section filter options, consider the following:

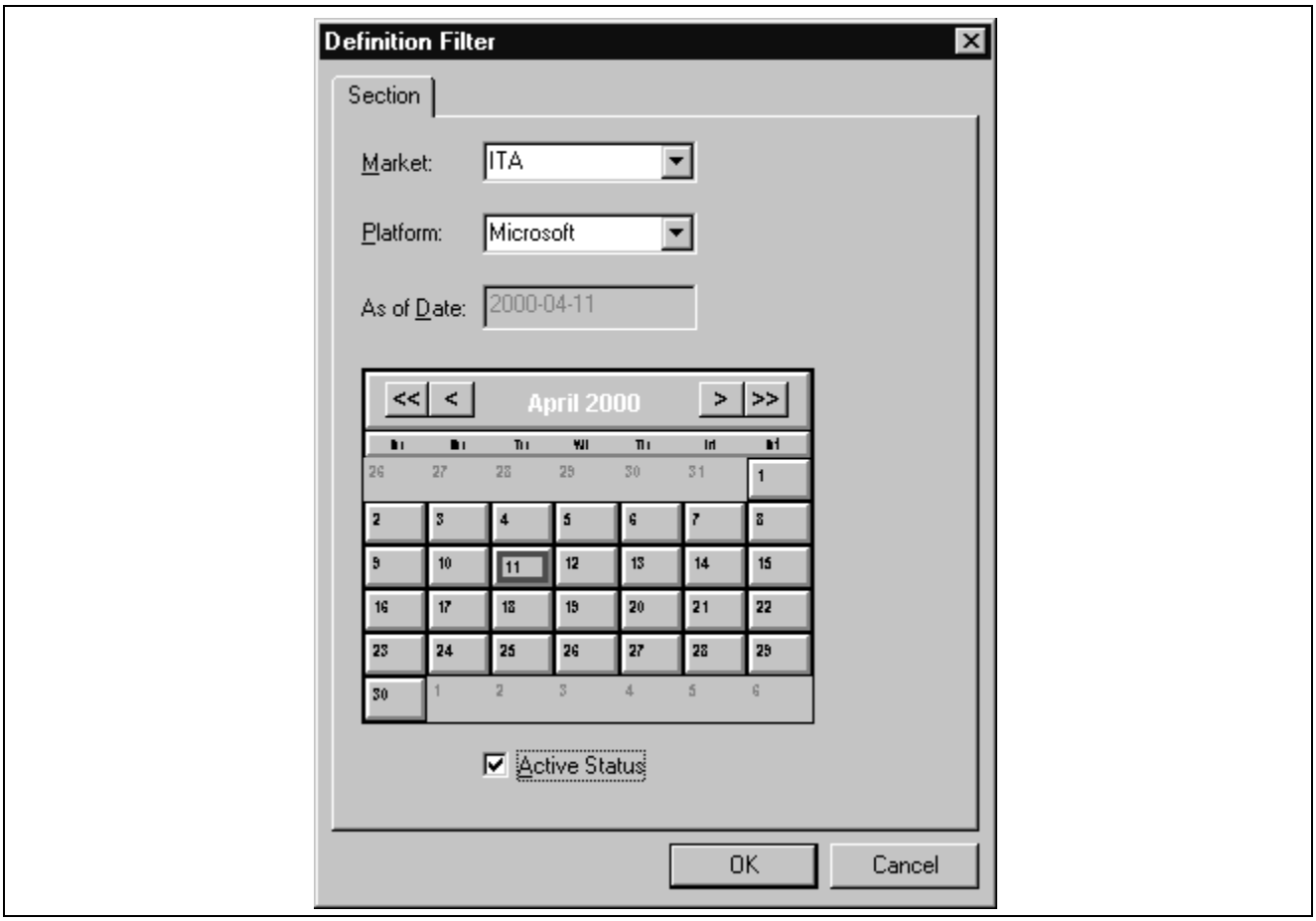
- The default is no filtering; therefore, all section definitions are included in this view.
- If you select custom filtering, the default filtering options are displayed while you're in the current session of PeopleSoft Application Designer.

  If you modify these filtering options and click OK, the new options are stored as the currently active options and the view is updated accordingly.
- If you select the default filter option, the original default options appear in the dialog box.

After clicking OK, the view reappears with only those sections that qualify. However, if you change the default options and do not click OK, these options are stored as a custom filtering request and the view reappears as necessary.

- If no platform-specific section is defined for the target filter value, the default (base platform) is always included, because this more accurately represents the PeopleSoft Application Engine runtime behavior.

If you select Section Filtering, Default, or Section Filtering, Custom, the following dialog box appears:



Definition Filter dialog box

In this example, only definitions that represent the following criteria appear in Definition and Program Flow views.

Market Select a market code to see only the definitions within that market. To see all market-related definitions for a program, you could update the default profile, or define a custom filter, selecting *(none)* from the Market drop-down list box. In the illustration, sections pertaining only to the Italian market (market code ITA) are shown.

Platform Select the platform filtering. In the illustration, sections that are defined only for the Microsoft SQL Server platform are shown. Select *Default* to display sections defined to be database-platform-independent (default platform). Specific platforms include DB2 UDB for OS/390 and z/OS, DB2/UNIX, Informix, Oracle, Microsoft, and Sybase.

As of Date	Select the date filtering. In the illustration, sections with an as of date equal to or greater than April 7, 2000 are shown. Select *None* to displays all sections regardless of effective date.
Active Status	Select to show active section definitions.

Note. All filtering options pertain only to section-level nodes.

Printing Program and Flow Definitions

You can print the program definition or program flow depending on which view you are in when you select print.

To print an Application Engine program definition:

1. Right-click and select Print in either Program Flow or Definition view, or select File, Print.
2. Select print options.

Program ID	Select to print the whole program.
Section	Select to print the currently selected section.
Step	Select to print the currently selected step (all steps are printed if Section is selected).
All Attributes	Select to print all detail level attributes for the specified node and its children.
SQL Statements	Select to print, for every SQL type action, the text of each SQL statement.
PeopleCode Statements	Select to print the text of the PeopleCode statements for every PeopleCode action.
XSLT Statements	Select to print any XSLT statements within a program.
Comments	Select to print the long description comments for the selected node and its children.
Include External Calls	Select to print the section detail of all external calls.
Max No. of Levels(maximum number of levels)	Specify the maximum number of recursive levels to print for the specified call sections.

Creating, Opening, and Renaming Programs

This section discusses how to:

- Create new programs.
- Open existing programs.
- Rename programs.

Creating New Programs

To create a new program definition:

1. Select File, New or press CTRL + N.
2. In the New dialog box, select *App Engine Program* from the Definition Type drop-down list box, and click OK.
3. Save and name your program.

   Select File, Save As, enter the name of your program in the Save Name As edit box, and click OK.

Note. It's also important to provide a program description and specify its owner in the properties dialog box for the new program.

See Chapter 3, "Creating Application Engine Programs," Setting Program Properties, page 21.

Opening Existing Programs

To open an existing program:

1. Select File, Open.
2. In the Open Definition dialog box, select *App Engine Program* from the Definition Type drop-down list box.
3. Enter your search criteria for the program you want, select your program in the search results list box, and click Open to open the program.

Renaming Programs

To rename a program:

1. Select File, Rename.
2. In the Rename dialog box, make sure that *App Engine Program* appears as the definition type.
3. In the box that contains the results of your search, click the program that you want to rename.
4. Click Rename.
5. Place the cursor in the box that appears around the highlighted program name.
6. Enter the new name for the program.
7. Click Rename again, and respond appropriately in the Confirm Rename dialog box.

Note. The system automatically modifies all static references in other programs to the renamed program. For instance, if you call the renamed program from another Application Engine program, the Call Section action in the calling program is modified to reflect the new program name. All sections and steps are saved under the new name. Only one occurrence of a program name can exist for a given database.

Note. If the renamed program is called in a dynamic Do action, the reference is not automatically modified. You should also manually check and modify any embedded references to the new program name in CallAppEngine or other PeopleCode functions.

Copying or Moving Program Elements

The following procedures apply to sections, steps, and actions. Note that when these functions are performed for a given object, the result applies not only to the selected object, but also includes its defined children, if they exist. Also note that all references to menu items apply not only to the main menu bar items, but also to their related items in the context menu, where applicable.

To copy a definition:

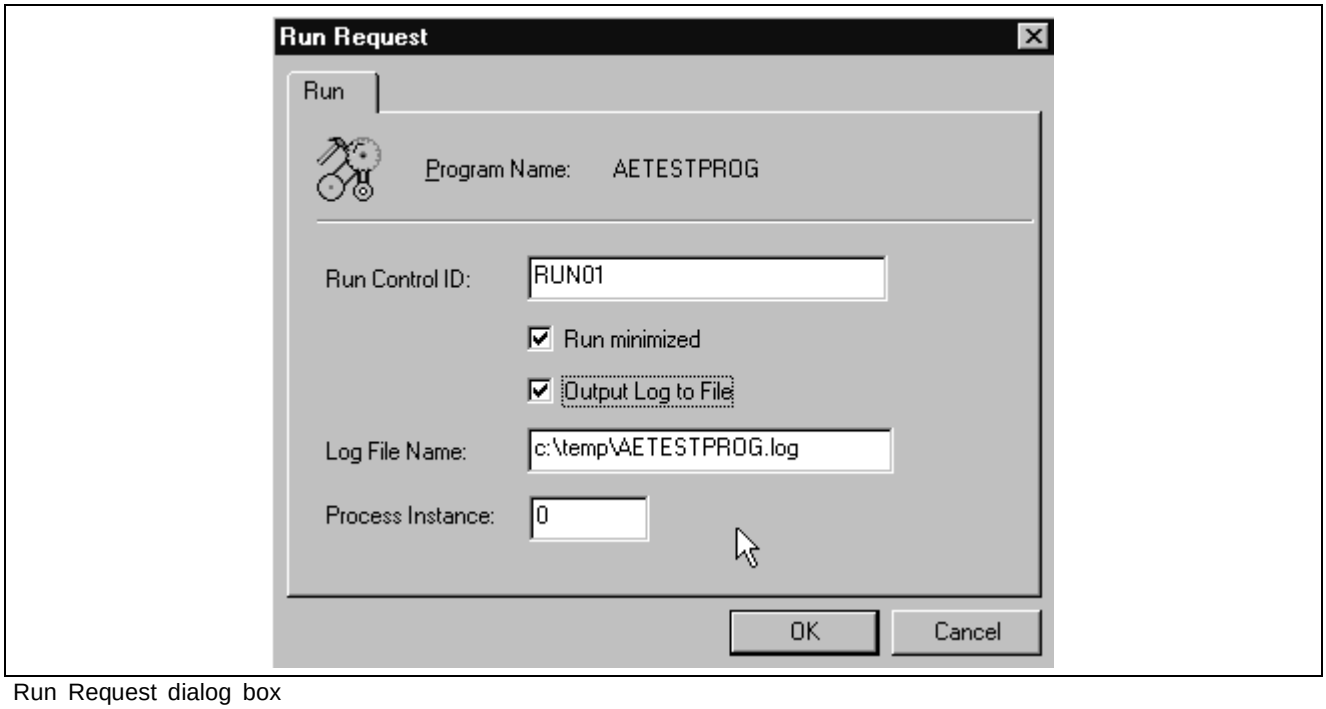
1. Select the definition.
2. Select Edit, Copy.
3. Position the cursor where you would like to put the copied definition, and select Edit, Paste.

To move a definition:

1. Select the definition object.
2. Select Edit, Cut.
3. Position the cursor at the target location and select Edit, Paste.

Testing Application Engine Programs

After creating or modifying your program, you can test it while in PeopleSoft Application Designer in two-tier mode. You use the Run Request dialog box:



Run Request dialog box

To run an Application Engine program in two-tier mode:

1. Select Edit, Run Program from the PeopleSoft Application Designer toolbar.

The Run Request dialog box appears.

2. Enter appropriate values.

   When you click OK, these values are passed as runtime parameters to the initiated PeopleSoft Application Engine runtime executable.

Run Control ID	Enter the run control ID of the program that you are testing.
Run Minimized	Select to have the window of the requested process minimized when it is submitted to run.
Output Log to File	Select to write the output log to a file.
Log File Name	Specify the log file name (enabled only when Output Log to File is selected).
Process Instance	Specify the process instance for this run request, or use the default value of zero if an instance number is not needed.

3. Click OK.

Setting Program Properties

This section discusses how to:

- Access properties.
- Set general properties.
- Set state record properties.
- Specify temporary tables.
- Set advanced properties.

Accessing Properties

When you have an Application Engine program open in PeopleSoft Application Designer, you can view and modify the properties assigned to an entire program just as you would a step or a section.

To view or modify the properties associated with a program, click the Properties button or select File, Definition Properties while the program is open. You can also press ALT+ENTER. The Program Properties dialog box appears.

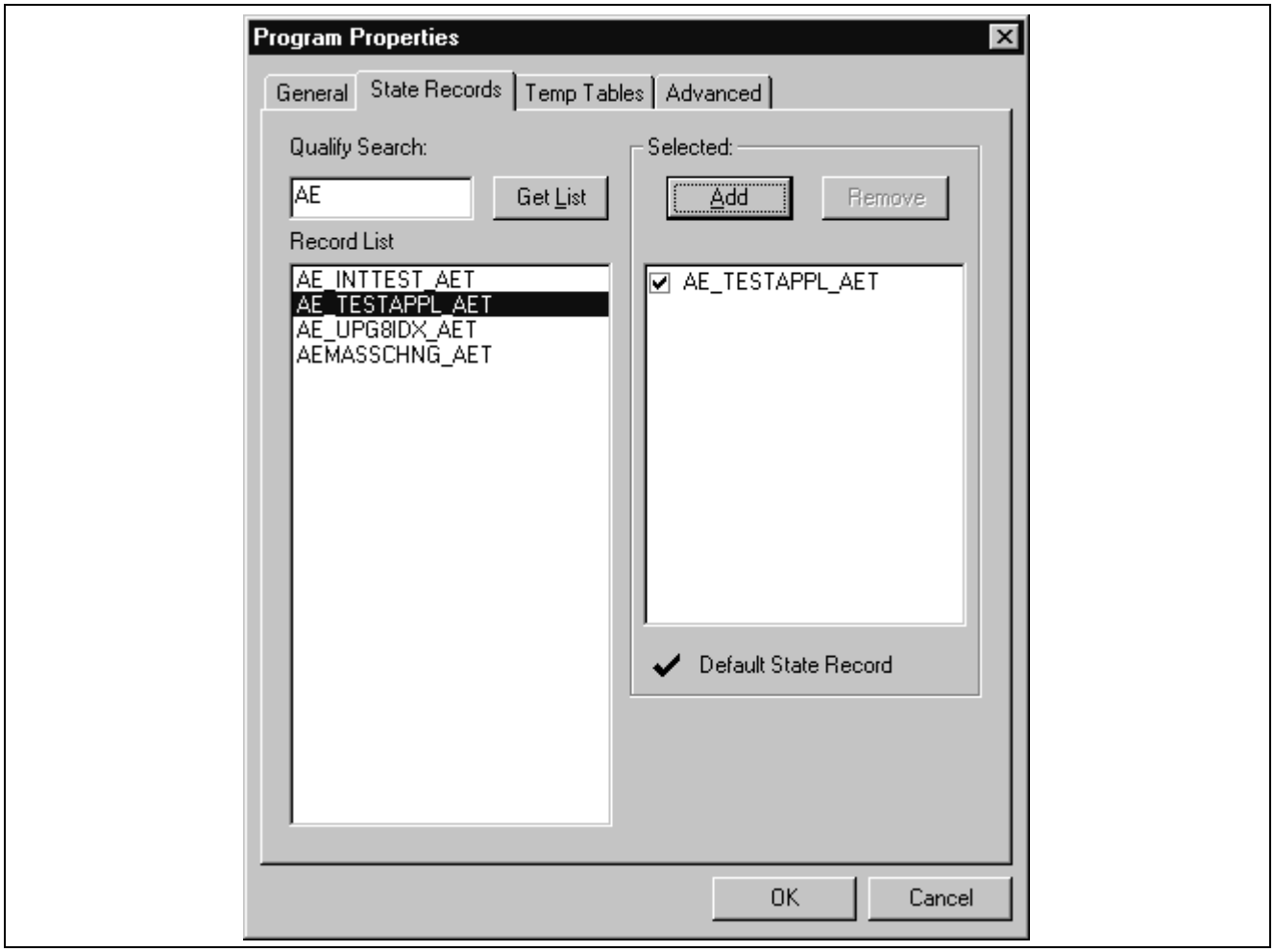
Setting General Properties

Access the Program Properties dialog box and select the General tab. You can specify identification values for your Application Engine program.

Owner ID	(Optional) Enter the owner ID for the program. The owner ID is a way to identify which definitions are owned by which PeopleSoft applications, such as PeopleSoft General Ledger, Accounts Receivables, and so on. The values in the drop-down list box are Translate table values associated with the OBJECTOWNERID field.

Setting State Record Properties

Select the State Records tab.



Program Properties dialog box: State Records tab

Qualify Search Enter any wildcard characters or complete table names to limit the results that appear in the record list. By default, the Record List box contains all record names that end with the extension AET. This extension identifies the record as an Application Engine record.

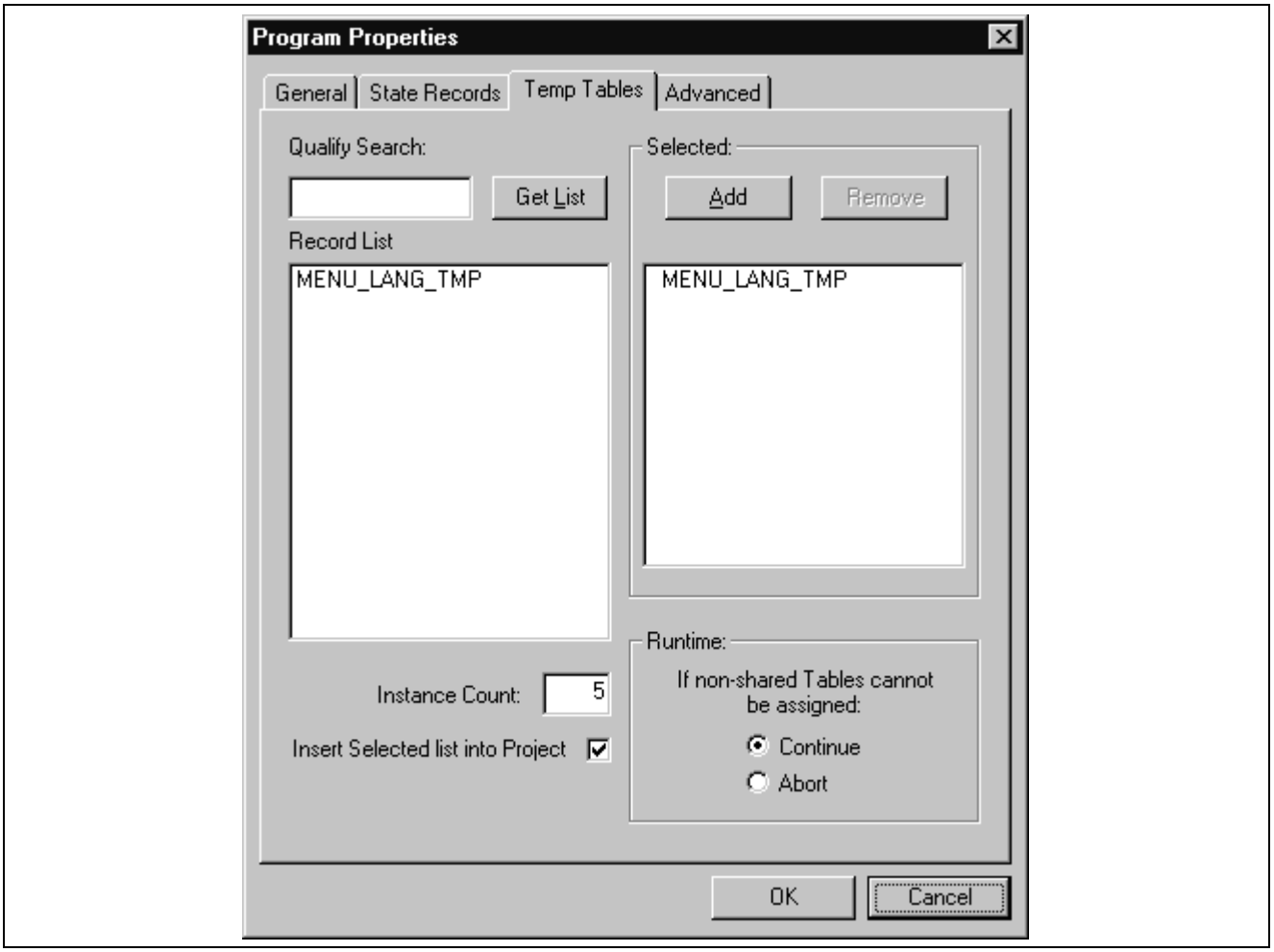
Get List Click to populate the Record List box.

Record List This text box contains the results of your state record search.

Selected Select state records for use with a particular program. Click Add to include selected records from the record list into the selected list. Click Remove to remove selected records from the selected list. Indicate which state record to act as the default state record by selecting its check box. For your default state record, you need to reference only fieldnames in your PeopleCode and SQL (for the active program). When you reference a non-default state record, you do so by using recname.fieldname.

Specifying Temporary Tables

Select the Temp Tables tab.



Program Properties dialog box: Temp Tables tab

Temporary tables store intermediate results during a program run.

Note. You must have already defined required temporary tables in your database prior to associating them with an Application Engine program.

Qualify Search Enter any wildcard characters or complete table names to limit the results that appear in the record list. By default, the Record List box contains only records that are of type Temporary Table. You apply this attribute when you create the record in PeopleSoft Application Designer.

Get List Click to populate the Record List box.

Record List This text box contains the results of your search for temporary tables.

Selected Select temporary tables for use with a particular program. Click Add to include selected records that appear in the record list. Click Remove to exclude selected records that appear in the selected list.

Instance Count Enter the number of physical tables to be created for each dedicated table for this program during the SQL Build procedure in PeopleSoft Application Designer. Typically, you would set this number to equal the maximum number of parallel program runs that you anticipate. For instance if you expect up to five instances of the same program to run simultaneously, then you would set the instance count to 5.

Insert Selected List into Project If the active Application Engine program definition belongs to a project, select to include the dedicated temporary tables for this program within the same project.

Runtime Control how an Application Engine program behaves if an instance of its specified dedicated temporary tables is not available. If you select Continue, then PeopleSoft Application Engine uses the base version, or nondedicated version, of the temporary tables. If you select Abort, then the program exits with an error message.

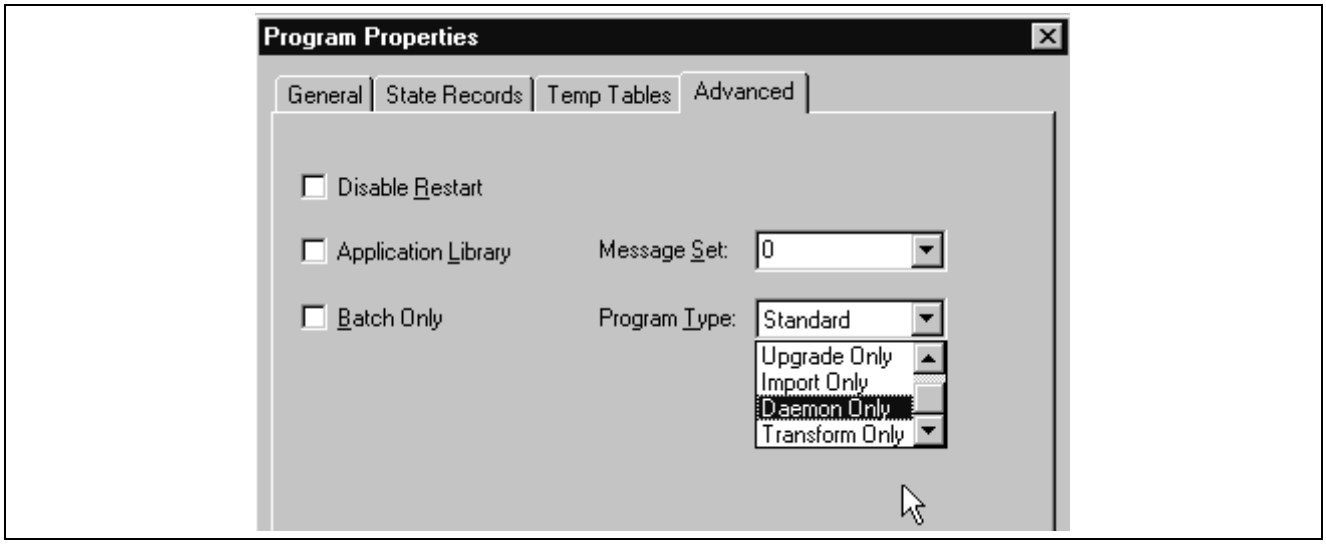
Note. If the table is keyed by PROCESS_INSTANCE, and the application SQL includes the process instance in the Where clause, then the table can be shared by multiple processes. The best performance, however, occurs when a program runs against a dedicated temporary table instance.

See Also

Chapter 9, "Using Temporary Tables," page 135

Setting Advanced Properties

Select the Advanced tab.



Program Properties dialog box: Advanced tab

Disable Restart Select to disable the built-in restart capabilities for a particular program.

Application Library In some cases, you may want a program to contain only a collection, or library, of common routines (in the form of callable sections) that you do not want to run as a standalone program. When sections are defined as public, other programs can call the sections, or routines, that exist in the library at runtime. Because this type of program is not designed to run as a standalone

program, it does not require the MAIN section, or initial entry point. Select this check box to rename or remove any existing MAIN section.

Note. An application library is the appropriate location to store a collection of shared Application Engine program sections. Libraries are not intended for storing a specific SQL action within a section. To share common SQL, use the SQL repository.

Batch Only Select for batch-only programs. Batch-only programs are not executed from the CallAppEngine PeopleCode function. Any dedicated temporary table used for batch-only programs do not have online instances created.

Message Set Specify the default message set value for this program. The system uses this message set value for all Log Message actions where the message set isn't specified.

Program Type Select from:

- *Standard:* Used by standard entry-point programs.
- *Upgrade Only:* Used by PeopleSoft upgrade utilities only.
- *Import Only:* Used by PeopleSoft import utilities only
- *Daemon Only:* Use for daemon-type programs.
- *Transform Only:* Support for XSLT programs.

Adding Sections

This section provides an overview of sections and discusses how to:

- Insert sections.
- Locate sections.
- Set section properties.

Understanding Sections

A section comprises steps and is somewhat equivalent to a COBOL paragraph or a PeopleCode function. You can create sections that are platform-independent or platform-specific, intended for a particular market, and effective-dated.

Whenever you create a new program, you simultaneously create a section called MAIN. The MAIN section identifies the program's entry point so that it can be called by another program.

Section Execution Order

A section is unique based on the program and section names, and based on its intended database platform and effective date. You can also create unique market-specific sections. When you execute an Application Engine program, it executes sections based on the following order of precedence:

1. If a section for the current market exists, execute it.

   Otherwise, execute the default GBL (global) market section.

2. If a section for the current platform, or database exists, execute it.

   Otherwise, execute the default database platform section.

3. If multiple effective-dated sections exists, execute the section with the most recent effective date, based on the current (run) date.

For example, suppose you have two versions of a particular section: SECT01 for the Public Sector market and SECT01 for the Global market. If you request to run the public sector version of the program, PeopleSoft Application Engine executes the Public Sector version of SECT01. If the program is running on Oracle, PeopleSoft Application Engine then looks for an Oracle version of the SECT01 for Public Sector.

Inserting Sections

To insert a section:

1. Select Insert, Section, or right-click and select Insert Section.

   The default name for a section that you insert is *Section N*, where *N* is an incremented number that attempts to provide a unique name for each section object. Unless you rename sections, the sections you add are named *SectionN+1*, where *N* is the last section you inserted. Consequently, you get names such as Section1, Section2, Section3, and so on.

   The designer inserts the new section directly beneath the subordinate objects within the highlighted object's owning section. For instance, if Section2 were selected, then Section4 would be inserted between Section2 and Section3, rather than after Section3.

   Note. Sections are always reordered alphabetically by name at save time to make it easier to locate a given section. However, order of execution is dependent on internal call section references, and is therefore independent of the order that sections are inserted and displayed.

   ---

2. Enter the remaining section property values.
3. Save the program.

Locating Sections

There are various methods to locate references to sections within an entire database as well as within a program.

Finding Call Section References

You can generate a list of all the references to a particular section. The list applies only to Application Engine programs defined within a single database.

To locate section references:

1. Open the program containing the shared, or called, section.
2. Select Edit, Find References.
3. The Find Definition References dialog box appears.
4. On the Call Sections tab, select the appropriate section from the Section name drop-down list box, or enter the name directly.

   By default, the current program name and MAIN section appears in the dialog box.

5. Click OK.
6. In the output window, view the generated list.

The output window lists the programs and sections that call a particular program. This list also shows the total call references made to a particular section. Call sections within the current program appear first in the list.

Double-click an item in the output window list to automatically navigate the definition view to that calling section.

Finding Sections Within the Current Program

Within large and more complicated Application Engine programs, such as those upgraded from a previous release, it is not uncommon to have over a hundred sections. Rather than scrolling through a large program, use the Go To Section feature.

Note. This feature applies only to the current program.

To automatically navigate to a selected section:

1. Select Edit, Go To Section.

   The Find Definition References dialog box appears.

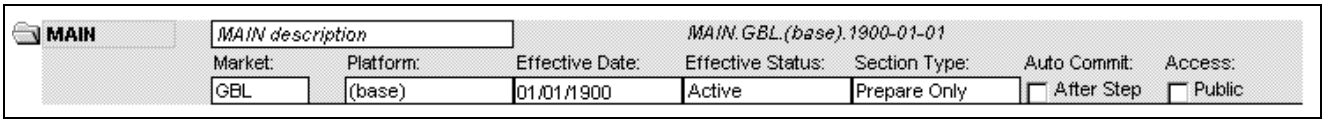
2. On the Go To Section tab, select the appropriate section from the Section name drop-down list box, or enter the name of the section.

3. Click OK.

   The Definition view scrolls to the first occurrence of the section with the name you selected.

Setting Section Properties

Controls that specify section properties are located in Definition view. For example, for each section included in your program, there is a node, as shown in the following example, from which you specify all of the attributes to associate with a particular section.



Section object

The values you specify at the section level generally apply to all the objects contained within that section.

Section Name Develop a naming convention and be consistent throughout your projects. You are limited to eight characters.

Market Select the market for which the section is intended. If a particular market is irrelevant to your batch program, keep the default market value of Global (GBL).

Platform Select the target database platform for which this section definition is to execute. Leave the default value for all sections whose defined actions are not specific to any given database platform.

Effective Date To make a particular section effective-dated, enter the target date.

Effective Status Specify whether a section is active, or enabled at runtime.

Section Type In the case of an abnormal termination of the program, the value of this system field specifies whether you must restart the section.

If a section controls a procedure that, if not run to completion, could corrupt or desynchronize your data, select *Critical Updates.* Otherwise, use the default value of *Preparation Only.*

Auto Commit Select to specify the commit level for the section. You can have no commit or you can have PeopleSoft Application Engine commit after the step successfully completes.

Public Select to enable a section to be called from another program.

Adding Steps

A step represents the smallest unit of work that can be committed in a program. When you create a program, you have a default MAIN section and step, initially named Step01.

This section discusses how to:

- Insert steps.
- Set step properties.

Inserting Steps

To insert a step:

1. Highlight the section or step that you want to precede the new step.

   For example, if you want the new step to be the first step in the section, select the section node. Otherwise, select the existing step that you want the new step to follow.

   Note. The name of the section in which you insert the step appears to the right of the step description. In large programs, this enables you to determine the section in which a step resides, if the section is not in view. Also, note that a sequence number appears on each step (001, 002, 003, and so on) so that you can determine a step's order within a section. The sequence numbering for steps begins at 001 within each section.

2. Select Insert, Step/Action.

   By default, the steps are given a default name of *StepN+1* beginning with *Step01*. Rename the step to better define the type of actions this step contains.

   Note. The designer continues to increment the step name until it has a unique step name within a section. If the designer is unable to create a unique name after 50 attempts, a new step is not inserted.

3. Specify a step name and the remaining values.

   To rename the step name, position the cursor in the step name edit box and enter a custom name. Only accept the default name for building quick, simple programs and for training purposes.

Setting Step Properties

You set step properties in Definition view.

Step Name Enter a name (up to eight characters).

Commit Specify the commit level for the step:

- *Default:* Select to inherit whatever commit level you specified for the section in which the step resides.
- *Later:* Select to postpone the commit until a subsequent commit occurs. Here you can override the section-level commit, if it happened to be set to *After Step.*
- *After Step:* Select if you have a commit level of *None* specified at the section level. This way you can override the section-level commit and commit a specific step within a section with no other commits.

Frequency Enabled only when a step contains one of the following actions: Do While, Do Select, or Do Until. Enter the numeric frequency with which PeopleSoft Application Engine should commit. If non-zero, PeopleSoft Application Engine commits every *N* iterations, and then again after the last iteration.

On Error Specify how PeopleSoft Application Engine should respond to an error at the step level. The On Error routine behaves the same for both SQL and PeopleCode actions. The program only terminates on errors, not warnings. Select from:

- *Abort:* The application terminates with an error message.
- *Ignore:* The program continues but logs an error message.
- *Suppress:*The program continues and presents no error message.
- *SQL:* Usually a program terminates if a SQL Prepare statement or execute fails. If you select *Ignore* or *Suppress*, errors on executing programs are suppressed, but errors on compiles still cause the program to terminate. Thus, if you select to reuse on an Update statement, the program fails on the compile if the SQL is incorrect, but it does not fail on a duplicate key error or similar error when the program executes.
- *PeopleCode:* There is a PeopleCode error in the program if the return code satisfies the statement `If (nRet & PCM_ERROR)`.

Status Select to activate a step. If the step is currently applicable to your program (and working) you'll probably want to keep it active.

Note. The On Error property does not apply to compile errors (for example, specifying erroneous SQL statements). It checks only for execution-type errors. If your program has a syntax error, the program terminates.

Specifying Actions

This section provides an overview of actions and discusses how to:

- Insert actions.
- Set action properties.
- Specify SQL actions.
- Specify Do actions.
- Specify PeopleCode actions.
- Specify Call Section actions.

- Specify Log Message actions.
- Specify XSLT actions.

Understanding Actions

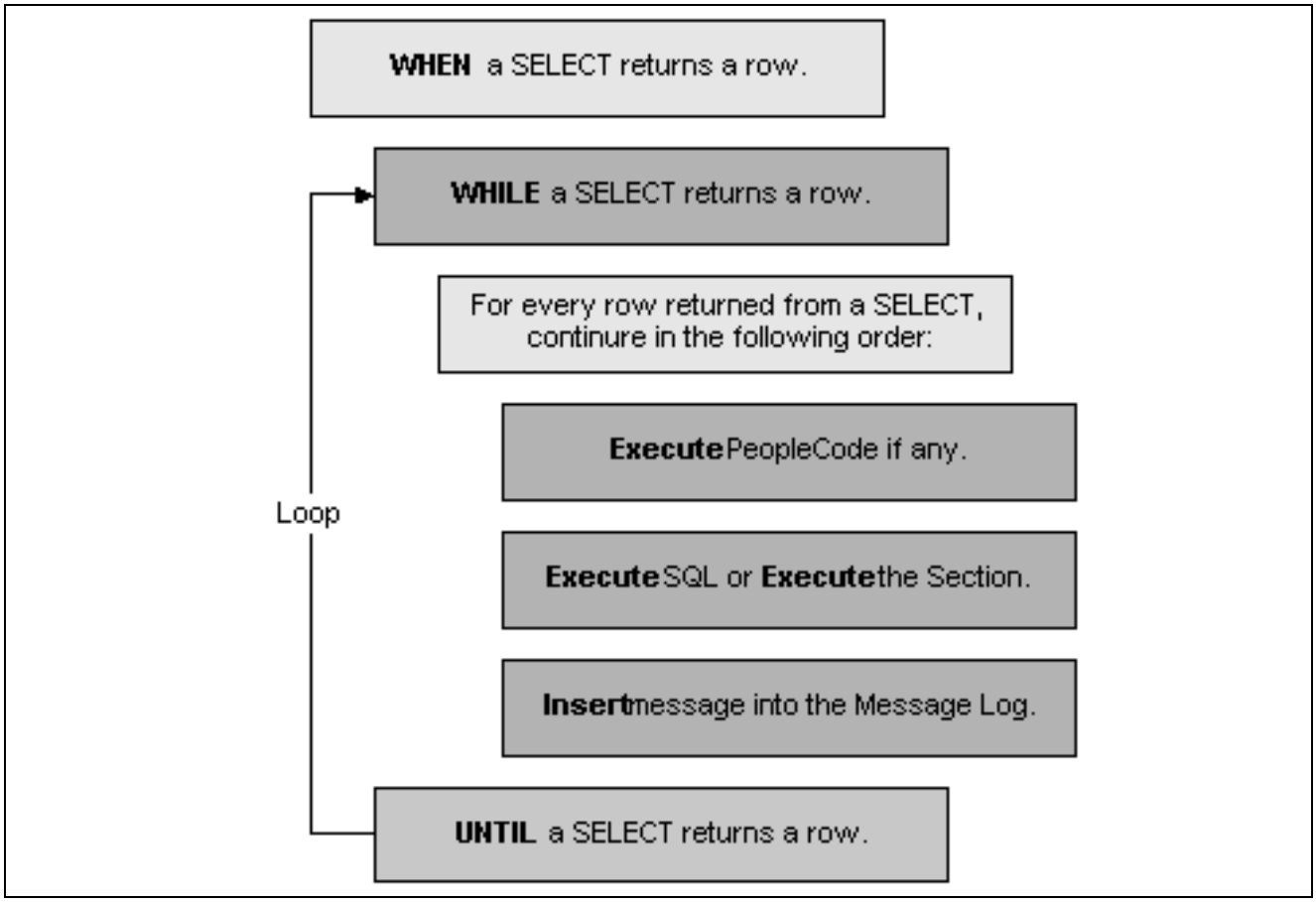
There are eight types of actions that you can include within a step, and a step can contain multiple actions. The actions you define for a step depend on the results that your program requires at each stage of execution.

The only mutually exclusive actions within a single step are Call Section and SQL Statement; you cannot add a Call Section action to a step that already contains a SQL Statement action and visa versa. You can include only one of each action type within a single step. Because there are eight types of actions, and two are mutually exclusive, the maximum number of actions a single step can contain is seven.

Action Execution Order

At runtime, the system evaluates actions by type and executes them within a strict hierarchy. For example, if both a Do When and PeopleCode action exist within a given step, PeopleSoft Application Engine always executes the Do When first.

The following diagram depicts the sequence and level of execution for each type of action:



Action execution hierarchy

As you add actions to a step in the definition view, the actions are initially inserted after the selected definition (the owning step or a prior action). However, following a save request or a refresh of the view, the designer reorders all actions to match the execution hierarchy. This feature helps you visualize the sequence in which each step of your program logic executes.

Note. A SQL action and a Call Section action are interchangeable and mutually exclusive. Only one of these two actions can appear within a step.

Remember the following when inserting actions:

- You cannot have more than one action of a specific type within the same step.
- You cannot have a SQL action and a Call Section action within the same step.
- You can only define XSLT type actions for programs defined as Transformation types (see the program properties).

Inserting Actions

To insert an action:

1. Highlight the step in which you want to insert the action.
2. Insert the action.

   You do this using one of the following methods:

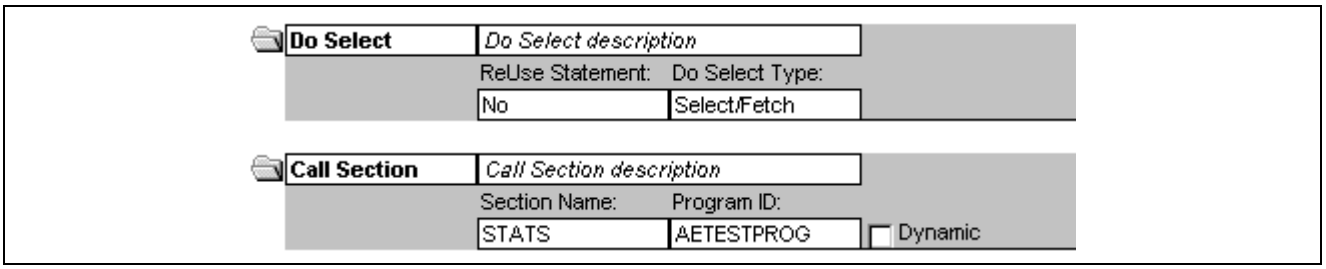
   - Select Insert, Step/Action.
   - Right-click the step and select Insert Step/Action.
3. Select the action type from the drop-down list box, or when current action type is selected, type the first character or so of the desired action type, then press TAB. The first (or only) type qualified by your entry is updated in this control.
4. Enter a description of the action.
5. Specify the appropriate properties for the action you selected.

Setting Action Properties

To modify action properties, you must have Definition view active. Because you can include a variety of actions within a step, there are different sets of properties specific to a particular action type. Depending on the action type you select, the properties that appear change.

For example, you can specify the reuse feature with a SQL action. This feature doesn't apply to a PeopleCode action; instead you need to specify how to respond to the PeopleCode program's return value.

The following illustration depicts how you can select action-specific properties for different action types.



Actions and associated properties

PeopleCode and all SQL action types invoke the related PeopleTools Editor to define or maintain the related text.

ReUse Statement Property

The ReUse Statement property is available for all SQL action types (SQL, Do When, Do While, Do Until, Do Select). You use the ReUse Statement property to optimize the SQL in your batch program. ReUse Statement converts any %BIND references to state record fields into real bind variables (:1, :2, and so on), enabling the Application Engine runtime process to compile the statement once, dedicate a cursor, and then re-execute it with new data as often as your program requires. When you are using SQL or a Do action to process a large volume of rows, one at a time, inside a fetch loop, compiling each statement that you issue can be a considerable performance issue. ReUse Statement is a way to combat potential performance decreases.

Note. You can have PeopleSoft Application Engine recompile a reused statement by using the %ClearCursor function.

When setting the ReUse Statement option, choose between these values:

Bulk Insert	When used in conjunction with statements like `INSERT INTO tablename (field1, field2...) VALUES (%BIND(ref1), %BIND(ref2)`, the Bulk Insert feature offers the most powerful degree of performance enhancements related to the ReUse Statement feature. This option turns on ReUse Statement, and, in addition, it holds all the data in a buffer and performs an insert only after a large volume of rows has gathered in the buffer. The number of rows allowed to gather in the buffer depends on your database platform. Storing data in the buffers is applicable only if you've selected *Bulk Insert* and the SQL is an Insert statement. For statements other than Insert, the Bulk Insert option is ignored.
No	Select this option to disable ReUse Statement. With ReUse off, the Application Engine runtime process recompiles the SQL statement every time the loop executes. By default, ReUse Statement is disabled.
Yes	Select this option to enable basic ReUse Statement functionality.

Note. The ReUse Statement property can offer significant performance gains. However, do not use it if %BIND variables are building parts of the SQL statement or are in the field list of a Select statement (this does not apply if you use the Static option in %BIND).

Specifying SQL Actions

This is the default action type for the first action within a given step. Use this action to perform the following SQL commands on multiple rows:

- Update
- Insert
- Delete
- Select

Note. Before you insert SQL (select View, SQL) into a SQL action within a new Application Engine program, you must have previously saved the program. This is required because the program name you use to save this definition is used to relate your program with the SQL objects you are about to create. The same is true for inserting PeopleCode.

With a SQL action, you use the SQL Editor to create and modify a SQL statement. Following are some examples of SQL statements:

```
%Select(AF_PERFM_AET.PREV_ASOF_DT)
SELECT %DateOut(ASOF_DT)
FROM PS_AF_FCST_SCHT%Bind(EPM_CORE_AET.TABLE_APPEND,NOQUOTES)
WHERE AFDEFN_ID = %Bind(AF_CORE_AET.AFDEFN_ID)
AND ASOF_DT = (SELECT MAX(ASOF_DT)
FROM PS_AF_FCST_SCHT%Bind(EPM_CORE_AET.TABLE_APPEND,NOQUOTES)
WHERE AFDEFN_ID = %Bind(AF_CORE_AET.AFDEFN_ID)
AND ASOF_DT < %Bind(AF_PERFM_AET.ASOF_DT))
```

Note. If you intend to include multiple SQL statements within a single action, you should use the meta-SQL construct %EXECUTE. The previous sample SQL statement sample contains bind variables from a previous Application Engine action.

No Rows Property

In addition to the ReUse Statement property, the No Rows property is available for SQL actions. If the SQL (Insert, Update, or Delete) associated with the SQL action does not return any rows, you must specify what the Application Engine program should do.

For example, you could use this in a case where you insert into a temporary table, and then you intend to perform further operations on the inserted rows (provided that some rows meet the criteria). If the initial combination of Insert and Select statements provides no rows, you could save the program from having to reselect on the temporary table prior to executing another operation, or you could also prevent the program from performing set operations on the table when there won't be any qualifying rows.

When you set the No Rows property, you choose from the following values:

Abort	The program terminates.
Section Break	PeopleSoft Application Engine exits the current section immediately, and control returns to the calling step.
Continue	The program continues processing.
Skip Step	PeopleSoft Application Engine exits the current step immediately and moves on to the next step. Application Engine ignores the commit for the current step at runtime. If the current step contains only one action, use Skip Step only to bypass the commit.

Note. Using No Rows in conjunction with a Truncate Table operation is unreliable. Some database platforms report zero rows affected for truncations, regardless of how many rows were in the table.

Specifying Do Actions

There are four types of PeopleSoft Application Engine actions that, although distinct from the others, can be grouped together as:

- Do When
- Do While
- Do Until

- Do Select

Use these actions to control the execution of your program. With these action types, you can control the execution of subsequent sections, actions, or SQL statements depending on the results of a Do SQL statement in the form of a Select statement. If you were coding in COBOL, you would perform similar actions using the If and While functions.

Any of the Do actions can control the execution of a section, a SQL statement, a PeopleCode program, or a log message. For example, a Do Select can execute a SQL statement for each row returned by the included Select statement.

Do When

When using a Do When action, consider the following:

- The Do When action is a Select statement that allows subsequent actions to be executed if any rows of data are returned.
- This action is similar to a COBOL If statement.

  A Do When statement runs before any other actions in the step. If the Do When statement returns any rows, the next action is executed. If the Do When conditions are not met, the remaining actions within that step are not executed. Your program executes a Do When action only once when the owning step executes.
- The only property that you can specify for the Do When action is the ReUse Statement property, which applies to all SQL-based actions.

Do While

The Do While action is a Select statement that, if present, runs before subsequent actions of the step. If the Do While does not return any rows of data, the action terminates. The Do While is identical to the COBOL While statement. Subsequent actions within the step are executed in a loop as long as at least one row is returned by the Select statement for the Do While action. If the Do While does not return any rows, the step is complete.

The only property that you can specify for the Do While action is the ReUse Statement property, which applies to all SQL-based actions.

Do Until

A Do Until action is a Select statement that runs after each action when a step completes. If the Select statement returns any rows of data, the step terminates.

- Use a Do Until action if you want the processing actions to execute at least once, and to execute over and over until a certain condition is true, such as until a Select statement returns some rows.
- You can also use a Do Until action to stop a Do Select action prematurely.

  For example, if the Select statement for the Do Until action does not return any rows, then the actions in the step are repeated (except if a Do When action appears in the step). Normally, a Do Select action continues until no rows are returned. If any rows of data are returned, the Do Select action stops and the step is not repeated.
- The only property that you can specify for the Do Until action is the ReUse Statement property, which applies to all SQL-based actions.

Do Select

The Do Select action is a Select statement that executes subsequent actions once for every row of data that the Do Select returns. For instance, a Do Select can execute a SQL statement for each row returned from the Select statement. The subsequent actions within the step are executed in a loop based on the results of the Select statement. The type of the Do Select determines the specific looping rules.

Like the other Do actions, for the Do Select action you can specify the ReUse Statement property, which applies to all SQL-based actions.

In addition to the ReUse Statement property, you must also specify another Do Select property: *Do Select Type*.

Note. PeopleSoft Application Engine does not commit a step containing a Do Select action with the Select/Fetch option enabled until the entire step completes successfully, regardless of the other options you have selected.

For example, suppose at the step level, you specified to commit every 100 iterations of the step. One of the actions of this step is a Do Select with Select/Fetch chosen. Because PeopleSoft Application Engine does not checkpoint or commit while the Do Select is active, the transaction performed by the actions within a step is not committed until the entire step completes successfully. This is also true if any sections are called from inside the loop.

Do Select Type Property

When you specify the Do Select Type property in a Do Select action, you choose from the following values:

Select/Fetch PeopleSoft Application Engine opens a cursor for the Do Select action, then, within that cursor, PeopleSoft Application Engine performs a Fetch statement for each iteration of the loop to get each row from the Select statement. When a Fetch statement results in an end of table message, the looping is complete. You can't restart this type of Select statement, because PeopleSoft Application Engine does not perform a checkpoint or a commit within the step containing this action while Select/Fetch is running. Ultimately, your program ignores the commit settings at runtime until the outermost Select/Fetch completes.

Note. When an Application Engine program is not set up for restartability, then commits are not controlled, monitored, or restricted by PeopleSoft Application Engine. When Restart is disabled, commits are controlled by the program.

Re-Select For each iteration of the loop, PeopleSoft Application Engine opens a cursor and fetches the first row. Your program processes the first row returned from the Select statement. The cursor is reopened for each iteration of the loop. With this type of Fetch statement, you typically want some aspect of the loop to eventually cause the Select statement to return no rows. Otherwise, there is no mechanism in place by which to exit the loop. This type of Do Select is restartable.

Restartable This option is similar to *Select/Fetch* in that PeopleSoft Application Engine opens the cursor associated with the Do Select action once, and then it performs a Fetch statement on each iteration of the loop to get each row from the Select statement. However, unlike the *Select/Fetch* option, you can restart this action, because PeopleSoft Application Engine performs a checkpoint in the middle of the step. PeopleSoft Application Engine treats this loop as if it is restartable, but it does not manage the restart. Make sure that the SQL you include within this action is such that, upon restart, the program recognizes where the previous run failed and where to restart processing. For example, you can employing a processed switch, or base the next Select statement on the key.

Specifying PeopleCode Actions

Use this action type to insert PeopleCode within your Application Engine program. You can invoke the PeopleCode Editor directly from the designer interface to code your PeopleCode programs.

With a PeopleCode action, there is only one property that you can specify—On Return.

Use the On Return value to determine how your Application Engine program reacts based on the return of your PeopleCode program. The On Return setting takes effect if your PeopleCode program issues a “return 1” or “exit 1.” You can use the True keyword in place of a non-zero numeric return.

When you specify the On Return property, you choose from the following values:

Abort	The program issues an error and exits immediately.
Break	The program exits the current step and section, and control returns to the calling step.
Skip Step	The program exits the current step, and continues processing at the next step in the section. If this is the last step in the section, the calling step resumes control of the processing.

Specifying Call Section Actions

Use the Call Section action to call another section defined in an Application Engine program. You can call a local section defined within your current program, and you can make external calls to a section defined in another Application Engine program.

The external section you intend to call must have its access property set to Public. If a section’s access property is set to Private, that section can be called only from within the same program. By default, a section’s access property is Private. If you attempt to make a call to a section that does not allow external calls, you receive an error message at runtime.

Note. You can call only programs that reside within the same database as the calling program.

Program ID Property

Because you can call sections defined in the current program or within external programs, you must first specify the program ID of the program containing the section you intend to call.

The default value is *(current).* If you call a section defined in another program, make sure that you first select the appropriate external program from the Program ID drop-down list box. The drop-down list box contains the names of all program definitions that currently exist in the database.

Section Name Property

Select from names defined in the program that appears in the Program ID list box. To call a section that is defined in an external program, select the program name in the Program ID edit box prior to selecting the section name.

Also use the Call Section action to call an entire external program. First select the program ID, then select section name MAIN. At runtime, this call executes the entire program defined by the value in the Program ID field.

Note. PeopleSoft Application Designer does not prevent you from calling the main section of the current program or the current section. For instance, Section1 can contain a step that has a local call section reference for Section1. This enables recursive calls, and should therefore be used with caution.

Dynamic Property

Use the AE_APPLID and AE_SECTION fields in the state record to execute different sections depending on the conditions a program encounters during runtime.

These two fields must be defined on the default state record for the program. If AE_APPLID is not present or is blank (at runtime), the current program is substituted for the AE_APPLID value. If AE_SECTION is not present or is blank, an error occurs.

When issuing a dynamic call, both the section and the program ID must be dynamically set. You enable a dynamic call by first having your program store different section names in the AE_SECTION field, and different program names in AE_APPLID field. The values you insert in these fields are normally based on various conditions met within your program. You then create a Call Section action that calls the section name defined in the state record field by selecting the Dynamic check box.

Selecting Dynamic automatically populates the AE_SECTION field with the symbolic value %Section, and the Program ID field with the symbolic value %AEAPPLID. At runtime, the program calls the section name stored in AE_SECTION that belongs to the program name defined by AE_APPLID.

Program Properties of Called Sections

When you call a section defined in an external program, the current program (the program containing the defined call section) defines the properties that apply to the running process. Suppose tracing is enabled for the current program, but tracing is disabled for the called program section. In this case, the called program has the trace option enabled at runtime because it inherits the calling program's properties.

For example, if program A calls program B, and program B calls program C, then the properties of A apply to both programs B and C. The calling program always controls the properties for the called program. In this case, program A controls the properties for program B, and because program B inherits the properties of program A, when program B calls program C, program A's properties also apply to program C.

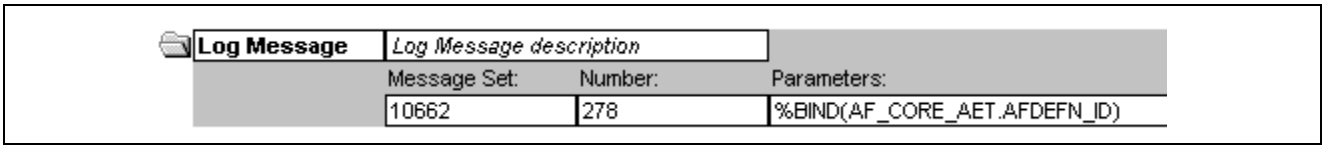
Note. Although program properties are inherited, state records do not follow this inheritance model.

State Records of Called Programs

When you call a program from another program, the called program's default state record becomes active until processing returns to the initial program. However, all of the state records associated with both programs are available. State records that are common between the two programs share values. To communicate between the two programs, or share %BIND variables, define the same state records in both programs.

Specifying Log Message Actions

Use this type of action to write a message to the message log. The message log refers to the PeopleTools table (PS_MESSAGE_LOG) where execution messages reside. Any substitution parameters are written to PS_MESSAGE_LOGPARM. The following illustration shows a Log Message action:



Log Message action

You can use the Log Message action to insert any type of messages. Typically, a Log Message action writes error messages to the message log, but you could also write informational or status messages.

Note. You can also use MessageBox PeopleCode to populate PS_MESSAGE_LOG instead of using the Log Message action. This enables you to easily record errors encountered within Application Engine PeopleCode programs.

Message Set and **Number** Select the message defined in the message catalog.

Parameters Enter values to insert in the log message. This field should be a comma-delimited list of values to substitute for the message variables (%1, %2, and so on) in the message text. These parameters can be hard-coded values or %Bind references. The information specified is inserted in the PS_MESSAGE_LOG at runtime, and any %Bind values are replaced by the current state record field values. You can then view the logged messages from the Process Monitor page.

For example, using message set 1012, number 10, the message reads "The total number of %1 rows exceeds the control count value, %2," and you need the following parameters:

Invoice, %Bind(CONTROL_CNT)

Suppose you run this program with the CONTROL_CNT field value of 120. When the process ends, the following message would be included on the Process Details dialog box in Process Monitor: "The total number of Invoice rows exceeds the control count value, 120."

Specifying XSLT Actions

These are used for transform programs only.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker, "Applying Filtering, Transformation and Translation," Using XSLT for Transformation

CHAPTER 4

Developing Efficient Programs

This chapter discusses how to:

- Use state records.
- Set commits.
- Reuse statements.
- Use Bulk Insert.
- Use set processing.

Using State Records

This section provides an overview of state records and discusses how to:

- Share state records.
- Choose a record type for state records.

Understanding State Records

You assign variables for your Application Engine program through state records, while sections, steps, and actions pass values to subsequent program steps through state records.

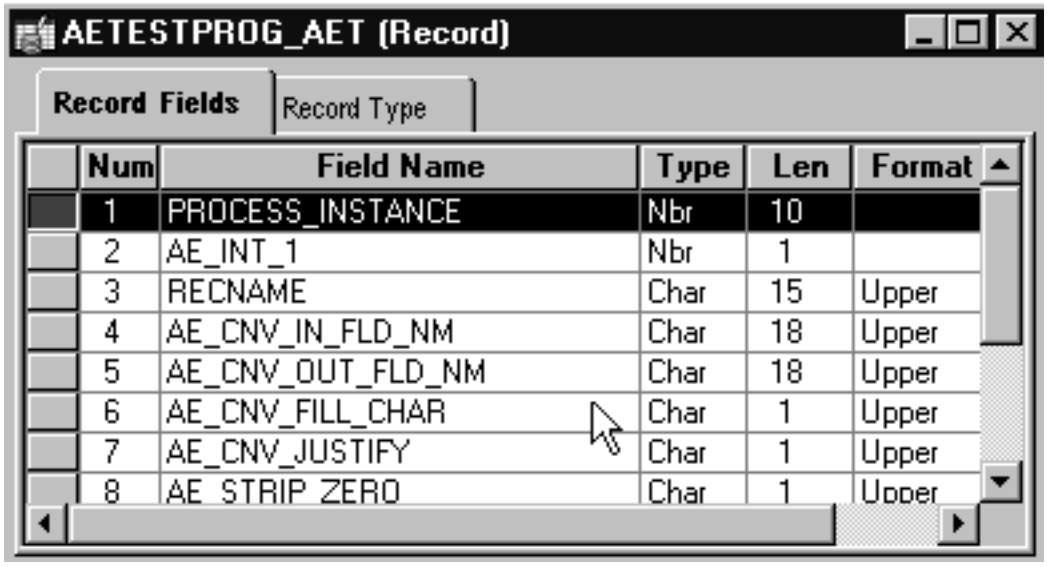
You can have any number of state records associated with a particular Application Engine program. However, only one record can be the default state record. You can specify both work (derived) and physical (SQL table) records to be used as state records. The only difference is that derived state records cannot have their values saved to the database at commit time, and so the values are lost during a restart. Therefore, PeopleSoft Application Engine erases the contents of derived state records at commit time if Restart is enabled for the current process.

A PeopleSoft Application Engine state record must have a process instance defined as the first field and the only key field, and the state record name must end with _AET.

Not all the database columns referenced in your program must be in the state record, just the columns that must be selected into memory so those values can be referenced in a subsequent program action. You may also want to include additional fields to hold pieces of dynamic SQL, to use as temporary flags, and so on.

PeopleSoft Application Engine supports long fields, unlike COBOL or Structured Query Reports (SQR). However, it allows only one long field per state record. You set a maximum size for the field in PeopleSoft Application Designer and make sure that the data space is compatible with the size of the field that you set.

PeopleSoft Application Engine also supports image fields and long text fields.



Sample state record

During batch processing, PeopleSoft Application Engine automatically performs all state record updates. When a program starts, it inserts a row into the state record that corresponds to the process instance assigned to that program run. PeopleSoft Application Engine updates the record whenever a commit operation occurs. When restart is enabled and a commit occurs, all state records that have been updated in memory are written to the database, except for derived state records, which are initialized instead.

After the program completes successfully, PeopleSoft Application Engine deletes the corresponding row in the state record. There is only one row in the state record for each process instance. Multiple programs can use the same state record, and each program has its own row based on the unique process instance key.

To set values in the state record, you use the %SELECT construct in a SQL statement or write PeopleCode that references the state field with the standard record.field notation. To reference fields in the state record, use the %BIND construct.

Sharing State Records

State records can be used by multiple sections and by multiple programs. When you call a section in another program, any additional state records defined for that program (as in state records that are not already in use by the calling program) are initialized, even if the program has been called previously during the run. However, state records that are common to both programs retain their current values.

To reference variables that exist within a state record, use the following:

```
%BIND(fieldname)
```

Unless a specific record name is specified preceding the fieldname, %BIND references the default state record. To reference a state record other than the default, use the following:

```
%BIND(recordname.fieldname)
```

In the case of a called program or section, if the called program has its own default state record defined, then PeopleSoft Application Engine uses that default state record to resolve the %BIND(fieldname). Otherwise, the called program inherits the calling program's default state record. In theory, the called program does not require a state record if all the fields it needs for processing exist on the calling program's state record.

For those state records that are shared between programs (during an external call section), any changes made by the called program remain when control returns to the calling program. Any subsequent actions in the calling program can access residual values left in the common state records by the called program. This can be useful to return output values or status to the calling program, yet it can also cause unforeseen errors.

Generally, a called program should not share state records with the caller unless you need to pass parameters between them. Most programs have their own set of state records unless a program calls another program that requires specific input or output variables. In that case, you must include the state record of the called program into the calling program's state record list, and make sure to set the input values before issuing the call section.

Choosing a Record Type for State Records

As a general rule, to preserve state record field values across commits in your program, you should store those values in a state record with a record type of SQL Table. Only derived/work-type state records store values that don't need to be accessed across commits. Derived/work records are, however, an excellent choice for temporary flags and dynamic SQL containers that are set and then referenced immediately. Because these values aren't needed later, you don't want to have to save them to the database at each commit. When you create your state record in PeopleSoft Application Designer, you should have an idea regarding how your state record will be used. With this information, you can select the appropriate record type to build.

With Application Engine programs, state records that are derived/work records function the same as SQL Table records. However, there is one notable distinction: unless you have disabled Restart, derived work records have their field values reinitialized after each commit. Therefore, unless you anticipate this behavior, you may encounter problems. One quick way to diagnose such a problem is to examine a trace. Typically, you see %BIND variables resolved to values prior to a commit, and then after the commit, they have no value.

This behavior is necessary to ensure consistency in the event of an abnormal termination and restart. During the restart, PeopleSoft Application Engine begins, or restarts, at the point of the last successful commit and restores the values of any state records with corresponding database tables. Derived/work records aren't associated with a physical database table, and consequently they can't be restored in the event of a restart.

Setting Commits

For new Application Engine programs that you develop, by default, the commit values at the section and the step level are turned off. No commits occur during the program run, except for the implicit commit that occurs after the successful completion of the program.

It's up to you to divide your program into logical units of work by setting commit points within your program. Typically, after PeopleSoft Application Engine completes a self-contained task, it might be a good time to commit. How often you apply commits affects how your program performs in the event of a restart. For set processing programs, commit early and often. For row-based processing, commit after every *N* iterations of the main fetch loop that drives the process.

If you have a step with a Do While, Do Until, or a Do Select action, you can set the frequency option, which drives your commit level. This enables you to set a commit at the step level that occurs after a specified number of iterations of your looping construct. Application Engine programs commit whenever they are instructed to do so, so you can enable the frequency option as well as have other individual commits inside of a loop.

The only restriction for batch runs occurs when you have restart enabled, and you are inside a Do Select action that is of the Select/Fetch type (instead of Re-select or Restartable). With Select/Fetch, all commits inside the loop are ignored, including the commit frequency if it's set.

The Restartable option is similar to Select/Fetch, except that you are implying to PeopleSoft Application Engine that your SQL is structured in such a way that it filters out rows that have been processed and committed. This enables a successful restart. One technique for accomplishing this is to have a processed flag that you check in the Where clause of the Do Select action, and you perform an update inside the loop (and before the commit) to set the flag to *Y* on each row that you fetch.

The commit logic is designed to perform a commit regardless of whether any database changes have occurred. The program commits as instructed, except when the program is restartable and at a point where a commit would affect restart integrity—inside a non-restartable Do Select action, for example.

When you set a step to commit by default, it means that the step's commit frequency is controlled by the section's auto commit setting. If the section is set to commit after every step, then the program commits. Otherwise, the program never commits unless the step is explicitly set to commit afterward.

Note. The Commit After, Later setting at the step level enables you to override the section setting if you don't want to commit after a particular step.

%TruncateTable Considerations

Some databases, such as Oracle, issue an implicit commit for a truncate command. If there were other pending (uncommitted) database changes, the results would differ if an abend occurred after the %TruncateTable. To ensure consistency and restart integrity, PeopleSoft Application Engine checks the following:

- Whether there are pending changes when resolving a %TruncateTable.
- If the program is at a point where a commit isn't allowed.

If either condition is true, PeopleSoft Application Engine issues delete from syntax instead.

Considerations with the No Rows Setting

The default for the No Rows setting (on the action) is *Continue.* This setting controls how your program responds when a statement returns no rows. In the case of %UpdateStats, you may want to set No Rows to *Skip Step* and thus skip the commit. For example, suppose you have a single Insert statement into a table, followed by an %UpdateStats. If the stats were current before the Insert statement, and the Insert statement affects no rows, then the %UpdateStats is unnecessary.

Reusing Statements

One of the key performance features of PeopleSoft Application Engine is the ability to reuse SQL statements by dedicating a persistent cursor to that statement.

Unless you select the ReUse property for a SQL action, %BIND fields are substituted with literal values in the SQL statement. The database has to recompile the statement every time it is executed.

However, selecting ReUse converts any %BIND fields into real bind variables (:1, :2, and so on), enabling PeopleSoft Application Engine to compile the statement once, dedicate a cursor, and re-execute it with new data multiple times. This reduction in compile time can result in dramatic improvements to performance.

In addition, some databases have SQL statement caching. Every time they receive SQL, they compare it against their cache of previously executed statements to see if they have seen it before. If so, they can reuse the old query plan. This works only if the SQL text matches exactly. This is unlikely with literals instead of bind variables.

When using ReUse, keep the following items in mind:

- ReUse is valid only for SQL actions.
- Use ReUse only if you do not use bind variables for column names.
- Use ReUse only if you have no %BIND variables in the Select list.
- If the SQL is dynamic, as in you are using %BIND to resolve to a value other than a standard bind value, and the contents of the bind change each time the statement gets executed, then you can't enable ReUse.

  In this situation, the SQL is different each time (at least from the database perspective) and therefore can't be reused.
- If you use the NOQUOTES modifier inside %BIND, there is an implied STATIC.

  For dynamic SQL substitution, the %BIND has a Char field and NOQUOTES to insert SQL rather than a literal value. If you enable ReUse, this means the value of the Char field gets substituted inline, instead of using a bind marker (as in :1, :2, and so on). The next time that the action executes, the SQL that it executes is the same as before, even if the value of a static bind has changed.
- To prepare a reused statement from scratch, because one of the static binds has changed and the SQL has to reflect that, use %ClearCursor.

If you are running DB2 on OS/390 or AS/400, use the ReUse property only when you are not using %BINDS as operands of the same operator, as shown in the following example:

```
UPDATE PS_PO_WRK1
SET TAX = %BIND(STATE) + %BIND(FED)
```

This causes error -417. You can modify the SQL so that you can use ReUse successfully. Suppose your program contains the following SQL:

```
UPDATE PS_PO_WRK1
   SET TAX = 0
   WHERE %BIND(TAX_EXEMPT) = %BIND(TAX_STATUS)
```

If you modify it to resemble the following SQL, ReUse works:

```
UPDATE PS_PO_WRK1
   SET TAX = 0
   WHERE %BIND(TAX_EXEMPT, STATIC) = %BIND(TAX_STATUS)
```

Using Bulk Insert

By buffering rows to be inserted, some databases can get a considerable performance boost. PeopleSoft Application Engine offers this non-standard SQL enhancement on the following databases: Oracle, Microsoft SQLServer, and DB2. This feature is named Bulk Insert. For those database platforms that do not support bulk insert, this flag is ignored.

You should consider using this feature only when an Insert SQL statement is called multiple times in the absence of intervening Commit statements.

PeopleSoft Application Engine ignores the Bulk Insert setting in the following situations:

- The SQL is not an Insert statement.
- The SQL is other than an Insert/Values statement that inserts one row at a time.

  For instance, the following statements are ignored: Insert/Select, Update, or Delete.

- The SQL does not have a Values clause.
- The SQL does not have a field list before the Values clause.

Note. Bulk Insert is also ignored when all three of the following conditions are true: the database platform is Oracle, the record contains and EFFDT field (effective date), and the record contains a mobile trigger. This is required because Oracle does not allow the reading of mutating tables in a row trigger.

In the situations where the Bulk Insert setting is ignored, PeopleSoft Application Engine still executes the SQL; it just doesn't take advantage of the performance boost associated with Bulk Insert.

To prepare or flush a Bulk Insert statement because one of the static binds has changed and the SQL has to reflect that, use %ClearCursor. A flush also occurs automatically before each commit.

Using Set Processing

This section provides an overview of set processing and discusses how to:

- Use set processing effectively.
- Avoid row-by-row processing.
- Use set processing examples.

Understanding Set Processing

Set processing is an SQL technique used to process groups, or sets of rows, at one time rather than processing each row individually. Set processing enables you to apply a business rule directly on the data (preferably while it resides in a temporary table) in the database using an Update or Insert/Select statement. Most of the performance gain is because the processing occurs in the database instead of loading the data into the application program, processing it, and then inserting the results back into the database tables. Because the data never leaves the database with set processing (whether it remains in the same table), you effectively eliminate the network round trip and database API overhead.

Note. Because the updates in set processing occur within the database, use temporary tables to hold transient data while your program runs. Although temporary tables are not required for set processing, they are often essential to achieve optimum performance in your batch program.

Using Set Processing Effectively

Keep the following in mind if you are developing new or upgrading older Application Engine programs to adhere to a set-based model.

SQL Expertise

Even if you're developing row-by-row programs with PeopleSoft Application Engine, you should be a SQL expert. With set-based programs, this is especially true. The following concepts are particularly important:

- Group by and Having clauses.
- Complex joins.
- Subqueries (correlated and non-correlated).

- Tools for your database to analyze complex SQL statements for performance analysis.

Typically, you use these SQL constructs to refine or filter the set to contain only the rows that meet particular criteria. Keep in mind that SQL is what you code with in PeopleSoft Application Engine, and Application Engine passes that SQL directly to the database, where it gets processed. If you have a complex SQL statement that works functionally, it may not necessarily perform well if it is not properly tuned.

Planning

Well-constructed, robust, and efficient Application Engine programs are usually the product of a detailed planning stage where loops, program flow, the use of temporary tables, sections, steps, and so on, are discussed.

In an ideal situation, address batch processing as a whole while you are designing the system. Sometimes, systems analysts and developers focus primarily on the online system during the database design, and then they consider the batch component within the existing database design. Set processing works best in an environment where the data models are optimized for set processing.

For example, you could have a separate staging table for new data that hasn't been processed, rather than having numerous cases where existing rows in a table get updated. In set processing, it is much easier to process the data after moving it to a temporary table using an Insert or Select statement rather than just using an update. Avoid performing updates on real application tables, and try to perform your updates on temporary tables. To minimize updating real application tables, structure your data model to prevent that.

Another important consideration is keeping historical data separate from active transactions. After the lifecycle of given piece of transaction data is over, so that no more updates are possible, consider moving that data to an archive or history table and deleting it from the real transaction table. This keeps the number of rows in the table to a minimum, which improves performance for queries and updates to your active data.

Temporary Tables

Although temporary tables are not required for set processing, well-designed temporary tables complement your set-based program in a variety of ways.

Creating temporary tables enables you to achieve one of the main objectives involved with set based processing—the processing remains on the database server. By storing transient data in temporary tables, you avoid the situation where the batch program fetches the data, row by row, and runs the business rule, processes the data, and then passes the updated data back to the database. If the program were running on the client, you encounter performance issues due to the network roundtrip and the diminished processing speed of a client compared to the database platform.

Your temporary tables should be designed to accomplish the following:

- Hold transaction data for the current run or iteration of your program.
- Contain only those rows of data affected by the business rule.
- Present key information in a denormalized, or flattened, form, which provides the most efficient processing.
- Switch the keys for rows coming from the master tables if needed.

  A transaction may use a different key than what appears on the master tables.

Denormalized Tables

The most efficient temporary tables store data in denormalized form. Because most programs need to access data that resides in multiple tables, it is more sensible to consolidate all of the affected and related data into one table, the temporary table. It's much more efficient for the program to run directly against the flattened temporary table rather than relying on the system to materialize complex joins and views to retrieve or update necessary data for each transaction.

If your program requires the use of a complex view to process transactions, then resolve the view into a temporary table for your program to run against. Each join or view that needs to materialize for each transaction consumes system resources and affects performance. In this approach, the system applies the join or view once (during the filtering process), populates the temporary table with the necessary information that the program needs to complete the transaction, and then runs the program against the temporary table as needed.

For example, consider the following situation:

A program needs to update 10,000 rows on the Customer table, which contains 100,000 rows of data. The Customer table is keyed by setID. To complete the transaction, the program references data that resides on a related table called PS_SET_CNTRL_REC. PS_SET_CNTRL_REC is used to associate setID and BUSINESS_UNIT values. The transaction is keyed by BUSINESS_UNIT.

Given that set of circumstances, the most efficient processing method would be similar to the following:

- Isolate affected or necessary data from both tables, and insert that into the temporary table.

  Now, instead of dealing with a 10,000-row Customer table and a join to a related table, the program faces a 10,000-row temporary table that contains all of the required data to join directly to the transaction data, which can also be in a temporary table. If all necessary columns reside on the temporary tables, the program can modify all the rows at once in a simple Update statement.

  This example presents two different uses of temporary tables. In one situation, the temporary table is designed to hold setup/control data in a modified form. In the other situation, the temporary table is designed to hold transaction data in a denormalized form, perhaps with additional work columns to hold intermediate calculations.

- Make sure the data appears in a denormalized form for optimum processing.
- Because the transaction is keyed by BUSINESS_UNIT, the temporary table that holds the control data should also be keyed by BUSINESS_UNIT.

  In this case, the table that holds the control data is the Customer table.

Avoiding Row-by-Row Processing

A set-based program is not an all-or-nothing situation. There are some rules that call for row-by-row processing, but these rules are the exception. However, you can have a row-by-row component within a mostly set-based program.

For example, suppose your program contains five rules that you'll run against your data. Four of those rules lend themselves well to a set-based approach, while the fifth requires a row-by-row process. In this situation, run the four set-based steps or rules first, and then run the row-by-row portion last to resolve the exceptions. Although it's not pure set-based processing, you still obtain better performance than if the entire program used a row-by-row approach.

When performing a row-by-row update, reduce the number of rows and the number of columns that you select to an absolute minimum to decrease the data transfer time.

For logic that cannot be coded entirely in set, try to process most of the transactions in set, and process only the exceptions in a row-by-row loop. A good example of an exception is the sequence numbering of detail lines within a transaction, when most transactions have only a single detail line. You can set the sequence number on all the detail lines to 1 by default, in an initial set-based operation, then execute a Select statement to retrieve only the exceptions (duplicates) and update their sequence numbers to 2, 3, and so on.

Avoid the tendency to expand row-by-row processing for more than what is necessary. For example, just because you're touching all of the rows of a given table in a specific row-based process, it doesn't mean that you gain in efficiency by running the rest of your logic on that table in a row-based manner.

When updating a table, it's OK to add another column to be set in the Update statement. However, do not add another SQL statement to your loop just because your program happens to be looping. If you can apply that SQL in a set-based manner, in most cases, you achieve better performance with a set-based SQL statement outside the loop.

The rest of this section describes techniques for avoiding row-by-row processing and enhancing performance.

Filtering

Using SQL, filter the set to contain only those rows that are affected or meet the criteria and then run the rule on them. Use the Where clause to minimize the number of rows to reflect only the set of affected rows.

Two-Pass Approach

Use a two-pass approach, wherein the first pass runs a rule on all of the rows, and the second pass resolves any rows that are exceptions to the rule. For instance, bypass exceptions to the rule during the first pass, and then address the exceptions individually in a row-by-row manner.

Parallel Processes

Divide sets into distinct groups, and then run the appropriate rules or logic against each set in parallel processes. For example, in terms of employee data, you could split the population into distinct sets of "hourly" and "salary," and then you could run the appropriate logic for each set in parallel.

Flat Temporary Tables

Flatten your temporary tables. The best temporary tables are denormalized and follow a flat file model for improved transaction processing.

For example, payroll control data might be keyed by set ID and effective dates rather than by business unit and accounting date. Use the temporary table to denormalize the data, and switch the keys to business unit and accounting date. Afterwards, you can construct a straight join to the Time Clock table, keyed by business unit and date.

Techniques to Avoid

Note the following:

- If you have a series of identical temporary tables, examine your refinement process.
- Don't attempt to accomplish a task that your database platform does not support, as in complex mathematics, non-standard SQL, and complex analytical modeling.

  Use standard SQL for set processing.
- Although subqueries are a useful tool for refining your set, make sure that you're not using the same one multiple times.

  If you are using the same subquery in more than one statement, you should probably have denormalized the query results into a temporary table. Identify the subqueries that appear frequently and, if possible, denormalize the queried data into a temporary table.

Using Set Processing Examples

The following sections each contain an example of set processing.

Payroll

In this example, suppose the payroll department needs to give a USD 1000 salary increase to everybody whose department made more than USD 50,000 profit. The following pseudocode enables you to compare the row-by-row and set-based approaches.

- Row-by-Row:

```
declare A cursor for select dept_id from department where profit > 50000;
open A;
fetch A into p_dept_id
while sql_status == OK
   update personnel set salary = (salary+1000) where dept_id = p_dept_id;
   fetch A into p_dept_id;
end while;
close A;
free A;
```

- Set-Based:

```
update personnel set salary = (salary + 1000)
   where exists
      (select 'X' from department
         where profit > 50000
         and personnel.dept_id = department.dept_id)
```

Note. The set-based example employs a correlated subquery, which is important in set-based processing.

Temporary Tables

One technique to improve database performance is to use a temporary table to hold the results of a common subquery. Effective dating and setID indirection are common types of subqueries that you can replace with joins to temporary tables. With the joins in place, you can access the temporary table instead of doing the subquery multiple times. Not only do most databases prefer joins to subqueries, but if you combine multiple subqueries into a single join as well, the performance benefits can be significant.

In this setID indirection example, you see a join from a transaction table (keyed by BUSINESS_UNIT and ACCOUNTING_DT) to a setup table (keyed by SETID and EFFDT).

To accomplish this using a single SQL statement, you need to bring in PS_SET_CNTRL_REC to map the business unit to a corresponding setID. This is typically done in a subquery. You also need to bring in the setup table a second time in a subquery to get the effective date (MAX(EFFDT) <= ACCOUNTING_DT). If you have a series of similar statements, this can be a performance issue.

The alternative is to use a temporary table that is the equivalent of the setup table. The temporary table is keyed by BUSINESS_UNIT and ACCOUNTING_DT instead of SETID and EFFDT. You populate it initially by joining in your batch of transactions (presumably also a temporary table) once, as described previously, to get all the business units and accounting dates for this batch. From then on, your transaction and setup temporary tables have common keys, which allows a straight join with no subqueries.

For the example, the original setup table (PS_ITEM_ENTRY_TBL) is keyed by SETID, ENTRY_TYPE and EFFDT.

The denormalized temporary table version (PS_ITEM_ENTRY_TAO) is keyed by PROCESS_INSTANCE, BUSINESS_UNIT, ENTRY_TYPE and ACCOUNTING_DT, and carries the original keys (SETID and EFFDT) as simple attributes for joining to other related setup tables, as in PS_ITEM_LINES_TBL for this example.

If the program references the setup table in only one Insert/Select or Select statement, you wouldn't see increased performance by denormalizing the temporary table. But if several SQL statements are typically executed in a single run, all of which join in the same setup table with similar setID and effective date considerations, then the cost of populating the temporary table up front provides long-term advantages.

- Original setup table version:

```
INSERT INTO PS_PG_PENDDST_TAO (...)
SELECT
. . . . .
 ( (I.ENTRY_AMT_BASE - I.VAT_AMT_BASE) *⇒
 L.DST_LINE_MULTIPLR * L.DST_LINE_PERCENT / 100 ),

 ( (I.ENTRY_AMT - I.VAT_AMT) *⇒
 L.DST_LINE_MULTIPLR * L.DST_LINE_PERCENT / 100 ),
. . . . .
FROM  PS_PENDING_ITEM I, PS_PG_REQUEST_TAO R, PS_ITEM_LINES_TBL L,
      PS_ITEM_ENTRY_TBL E, PS_SET_CNTRL_REC S, PS_BUS_UNIT_TBL_AR B
. . . . .

WHERE
 AND L.ENTRY_REASON = I.ENTRY_REASON
 AND L.SETID = E.SETID
 AND L.ENTRY_TYPE = E.ENTRY_TYPE
 AND L.EFFDT = E.EFFDT
. . . . .
 AND E.EFF_STATUS = 'A'
 AND S.RECNAME = 'ITEM_ENTRY_TBL'
 AND S.SETID = E.SETID
 AND S.SETCNTRLVALUE = I.BUSINESS_UNIT
 AND E.ENTRY_TYPE = I.ENTRY_TYPE
 AND E.EFFDT = (SELECT MAX(EFFDT) FROM PS_ITEM_ENTRY_TBL Z
                  WHERE Z.SETID = E.SETID
                    AND Z.ENTRY_TYPE = E.ENTRY_TYPE
                    AND Z.EFF_STATUS = 'A'
                    AND Z.EFFDT <= I.ACCOUNTING_DT )
 AND B.BUSINESS_UNIT = I.BUSINESS_UNIT
/
```

- Denormalized temporary table version:

```
INSERT INTO PS_ITEM_ENTRY_TAO
. . . . .
SELECT DISTINCT %BIND(PROCESS_INSTANCE), I.BUSINESS_UNIT, I.ACCOUNTING_DT,
 E.ENTRY_TYPE...
. . .
FROM  PS_PENDING_ITEM I, PS_PG_REQUEST_TAO R,
      PS_ITEM_ENTRY_TBL E, PS_SET_CNTRL_REC S, PS_BUS_UNIT_TBL_AR B
WHERE R.PROCESS_INSTANCE = %BIND(PROCESS_INSTANCE)
 AND R.PGG_GROUP_TYPE = 'B'
 AND I.POSTED_FLAG = 'N'
 AND R.GROUP_BU = I.GROUP_BU
```

```
 AND R.GROUP_ID = I.GROUP_ID
 AND E.EFF_STATUS = 'A'
 AND S.RECNAME = 'ITEM_ENTRY_TBL'
 AND S.SETID = E.SETID
 AND S.SETCNTRLVALUE = I.BUSINESS_UNIT
 AND E.ENTRY_TYPE = I.ENTRY_TYPE
 AND E.EFFDT =  ( SELECT MAX(EFFDT) FROM PS_ITEM_ENTRY_TBL Z
                   WHERE Z.SETID = E.SETID
                     AND Z.ENTRY_TYPE = E.ENTRY_TYPE
                     AND Z.EFF_STATUS = 'A'
                     AND Z.EFFDT <= I.ACCOUNTING_DT )
 AND B.BUSINESS_UNIT = I.BUSINESS_UNIT
/
INSERT INTO PS_PG_PENDDST_TAO (...)
SELECT ...
 ( (I.ENTRY_AMT_BASE - I.VAT_AMT_BASE) *⇒
 L.DST_LINE_MULTIPLR * L.DST_LINE_PERCENT / 100 ),
( (I.ENTRY_AMT - I.VAT_AMT) *⇒
 L.DST_LINE_MULTIPLR * L.DST_LINE_PERCENT / 100 ),

. . . . .
FROM  PS_PENDING_ITEM I, PS_PG_REQUEST_TAO R, PS_ITEM_LINES_TBL L,
      PS_ITEM_ENTRY_TAO E
. . . . .
WHERE
. . . . .
 AND L.ENTRY_REASON = I.ENTRY_REASON
 AND L.SETID = E.SETID
 AND L.ENTRY_TYPE = E.ENTRY_TYPE
 AND L.EFFDT = E.EFFDT
. . . . .
 AND E.BUSINESS_UNIT = I.BUSINESS_UNIT
 AND E.ACCOUNTING_DT = I.ACCOUNTING_DT
 AND E.ENTRY_TYPE = I.ENTRY_TYPE
/
```

Platform Issues

Set processing does not behave the same on every database platform. On some platforms, set processing can encounter performance breakdowns. Some platforms do not optimize update statements that include subqueries.

For example, environments that are accustomed to updates with subqueries get all the qualifying department IDs from the Department table, and then, using an index designed by an application developer, update the Personnel table. Other platforms read through every employee row in the Personnel table and query the Department table for each row.

On platforms where these types of updates are a problem, try adding some selectivity to the outer query. In the following example, examine the SQL in the Before section, and then notice how it is modified in the After section to run smoothly on all platforms. You can use this approach to work around platforms that have difficulty with updates that include subqueries.

Note. In general, set processing capabilities vary by database platform. The performance characteristics of each database platform differ with more complex SQL and set processing constructs. Some database platforms allow additional set processing constructs that enable you to process even more data in a set-based manner. If performance needs improvement, you must tailor or tune the SQL for your environment. You should be familiar with the capabilities and limitations of your database platform and can recognize, through tracing and performance results, the types of modifications you need to incorporate with the basic set processing constructs described.

- Basic version:

```
UPDATE PS_REQ_LINE
SET SOURCE_STATUS = 'I'
WHERE
EXISTS
(SELECT 'X' FROM PS_PO_ITM_STG STG
WHERE
STG.PROCESS_INSTANCE =%BIND(PROCESS_INSTANCE)  AND
STG.PROCESS_INSTANCE =PS_REQ_LINE.PROCESS_INSTANCE AND
STG.STAGE_STATUS = 'I'  AND
STG.BUSINESS_UNIT = PS_REQ_LINE.BUSINESS_UNIT AND
STG.REQ_ID = PS_REQ_LINE.REQ_ID AND
STG.REQ_LINE_NBR = PS_REQ_LINE.LINE_NBR)
```

- Optimized for platform compatibility:

```
UPDATE PS_REQ_LINE
SET SOURCE_STATUS = 'I'
WHERE
PROCESS_INSTANCE = %BIND(PROCESS_INSTANCE) AND
 EXISTS
(SELECT 'X' FROM PS_PO_ITM_STG STG
WHERE
STG.PROCESS_INSTANCE =%BIND(PROCESS_INSTANCE)  AND
STG.PROCESS_INSTANCE =PS_REQ_LINE.PROCESS_INSTANCE AND
STG.STAGE_STATUS = 'I'  AND
STG.BUSINESS_UNIT = PS_REQ_LINE.BUSINESS_UNIT AND
STG.REQ_ID = PS_REQ_LINE.REQ_ID AND
STG.REQ_LINE_NBR = PS_REQ_LINE.LINE_NBR)
```

Note. This example assumes that the transaction table (PS_REQ_LINE) has a PROCESS_INSTANCE column to lock rows that are in process. This is another example of designing your database with batch performance and set processing in mind.

This modification enables the system to limit its scan through PS_REQ_LINE to only those rows that the program is currently processing. At the same time, it enables a more set-friendly" environment to first scan the smaller staging table and then update the larger outer table.

CHAPTER 5

Using Meta-SQL and PeopleCode

This chapter provides an overview of PeopleSoft Application Engine meta-Structured Query Language (SQL) and discusses how to:

- Use PeopleCode in Application Engine programs.
- Include dynamic SQL.
- Use Application Engine meta-SQL.

Understanding PeopleSoft Application Engine Meta-SQL

Application Engine meta-SQL is divided into the following categories:

- Construct.

  A construct is a direct substitution of a value, which helps to build or modify a SQL statement.

- Function.

  A function performs an action on its own or causes another function to be called.

- Meta-variable.

  A meta-variable allows substitution of text within SQL statements.

Note. Some meta-SQL elements can be used only in Application Engine programs, some can be used both in Application Engine programs and in other environments, and some can't be used in Application Engine programs at all. Only meta-SQL elements that can be used in PeopleSoft Application Engine are discussed in this PeopleBook. You can find a complete reference to all PeopleSoft meta-SQL elements in the *PeopleTools 8.45 PeopleCode Reference PeopleBook*.

See Also

Chapter 5, "Using Meta-SQL and PeopleCode," Application Engine Meta-SQL Reference, page 66

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL"

Using PeopleCode in Application Engine Programs

This section provides an overview of PeopleCode and Application Engine programs and discusses how to:

- Decide when to use PeopleCode.

- Consider the program environment.
- Access state records with PeopleCode.
- Use If/Then logic.
- Use PeopleCode in loops.
- Use the AESection class.
- Make synchronous online calls to Application Engine programs.
- Use the file class.
- Call COBOL modules.
- Call PeopleTools APIs.
- Use the CommitWork function.
- Use PeopleCode examples.

Understanding PeopleCode and Application Engine Programs

Inserting PeopleCode within Application Engine programs enables you to reuse common function libraries and improve performance. In many cases, a small PeopleCode program used instead of Application Engine PeopleCode is an excellent way to build dynamic SQL, perform simple If/Else edits, set defaults, and perform other tasks that don't require a trip to the database.

Scope of Variables

The following table presents the different types of variables typically used in Application Engine programs and their scope.

Type of Variable	Scope	Comments
State record (work record)	Transaction (unit of work)	Using a work record as your Application Engine state record means that the values in the work record cannot be committed to the database. Commits happen as directed, but any values in work records are not retained after a commit.
State record (database record)	Application Engine program	Using a database record as your Application Engine state record preserves the values in the state record on commit, and the committed values are available in the event of a restart.
Local PeopleCode variables	PeopleCode program	Local PeopleCode variables are available only for the duration of the PeopleCode program that is using them.

Type of Variable	Scope	Comments
Global PeopleCode variables	Application Engineprogram	Global PeopleCode variables are available during the life of the program that is currently running. Any global PeopleCode variables are saved when an Application Engine program commits and checkpoints, and therefore they are available in the event of a restart.
Component PeopleCode variables	Application Engine program	Component PeopleCode variables act like global variables to PeopleSoft Application Engine.

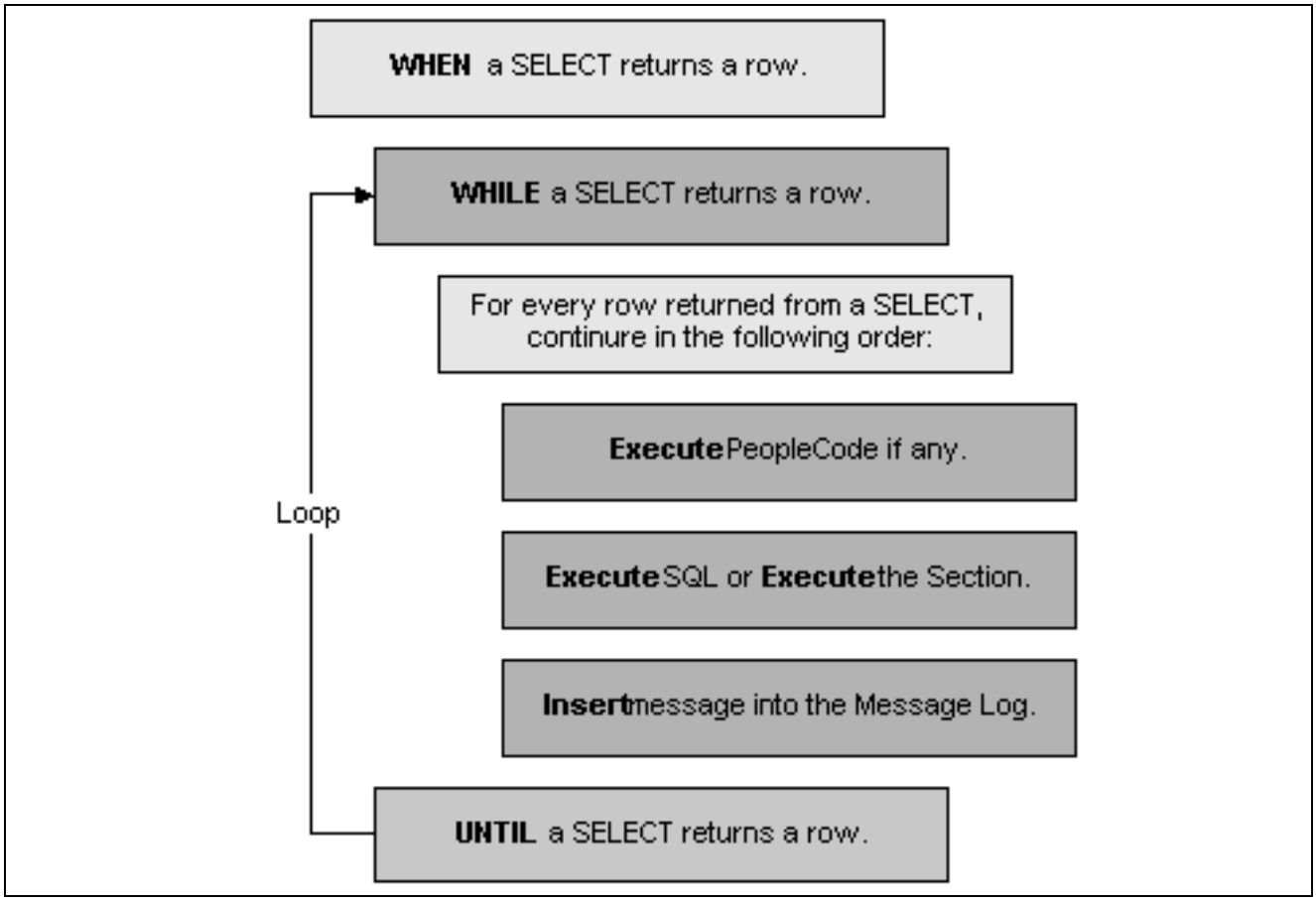
Action Execution Order

No other types of actions are required within a step in conjunction with a PeopleCode action (or program). So, you can have a step that contains nothing but one PeopleCode action. If you include other actions with your PeopleCode action within the same step, it's important to keep in mind the execution hierarchy.

With PeopleCode actions, Application Engine executes the PeopleCode program *before* the SQL, Call Section, or Log Message actions, but a PeopleCode program executes *after* any program flow checks.

Because there are multiple action types, they must execute in agreement within a system, and therefore the order in which action's execute is significant. At runtime, actions defined for a given step are evaluated based on their action type. All of the action types exist within a strict hierarchy of execution. For example, if both a Do When and PeopleCode action exist within a given step, the Do When is always executed first.

The following example shows the sequence and level of execution for each type of action:



Action execution hierarchy

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Application Engine

Deciding When to Use PeopleCode

PeopleSoft Application Engine is not intended to execute programs that include nothing but PeopleCode actions. PeopleSoft Application Engine's primary purpose is to run SQL against your data.

For the most part, use PeopleCode for setting If, Then, Else logic constructs, performing data preparation tasks, and building dynamic portions of SQL statements, while still relying on SQL to complete the bulk of the actual program processing. You also use PeopleCode to reuse online logic that's already developed. And, of course, PeopleCode is the tool for taking advantage of the new technologies, such as component interfaces and application classes.

Most programs need to check that a certain condition is true prior to executing a particular section. For example, if the hourly wage is less than or equal to X, do Step A; if not, fetch the next row. In certain instances, you need to modify variables that exist in a state record. PeopleCode enables you to set state record variables dynamically.

Considering the Program Environment

When writing or referencing PeopleCode in a PeopleCode action, you must consider the environment in which the Application Engine program runs. *Environment* indicates the differences between online and batch modes. Application Engine programs usually run in batch mode, and, consequently, your PeopleCode cannot access pages or controls as it can while running in online mode. Any PeopleCode operations that manipulate pages will not run successfully. Even if you invoke your Application Engine program online from a record or a page using the CallAppEngine PeopleCode function, the Application Engine PeopleCode still does not have direct access to component buffers.

Any record field references that appear in a PeopleCode action can refer only to fields that exist on an Application Engine state record. Component buffers, controls, and so on are still inaccessible even if you define the page records as state records on the Program Properties dialog box. An Application Engine program can access only state records or other objects you create in PeopleCode.

However, you do have several options for passing data from a component buffer to an Application Engine program. You can use the CallAppEngine PeopleCode function, or you can define global variables.

Passing Parameters Through the CallAppEngine Function

For individual page fields and simple PeopleCode variables, such as numbers and strings, you can use the CallAppEngine PeopleCode function to pass values as parameters.

To use the CallAppEngine function:

1. Declare a record object in PeopleCode.

   Here is an example:

   ```
   Local Record &MyRecord;
   ```

2. Assign the record objects to any state record that you want to pass to the Application Engine program.

   Record objects are parameters to the CallAppEngine function.

3. Set the appropriate values on that state record.
4. Include the record object in the function call.

After these values get set in the state record, all the actions in a particular program can use the values, not just the PeopleCode actions.

Defining Global Variables

You can also define global variables or objects in PeopleCode before calling an Application Engine program. Application Engine PeopleCode actions only are able to access the variables you define; however, the PeopleCode could set a state record field equal to a number or string variable for use by other Application Engine actions.

Also, an Application Engine PeopleCode program can read or update a scroll area or a grid using a global rowset object. When accessing a scroll area or a grid from Application Engine PeopleCode, the same rules apply, and the same illegal operations are possible that you see with accessing PeopleCode not in an Application Engine program.

The parameters submitted in a CallAppEngine are by value. These parameters seed the specified Application Engine state record field with the corresponding value. If that value is changed within PeopleSoft Application Engine by updating the state record field, the component data is not be affected. The only way to update component buffers or external PeopleCode variables from PeopleSoft Application Engine is to use global PeopleCode variables and objects.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CallAppEngine

Accessing State Records with PeopleCode

Executing PeopleCode from Application Engine steps enables you to complete some simple operations without having to use SQL. For example, to assign a literal value to an Application Engine state record field using SQL you may have issued a statement similar to the following:

```
%SELECT(MY_AET.MY_COLUMN)
SELECT 'BUSINESS_UNIT' FROM PS_INSTALLATION
```

You can use a PeopleCode assignment instead:

```
MY_AET.MY_COLUMN = "BUSINESS_UNIT";
```

Similarly, you can use a PeopleCode If statement instead of using a Do When action to check the value of a state record field.

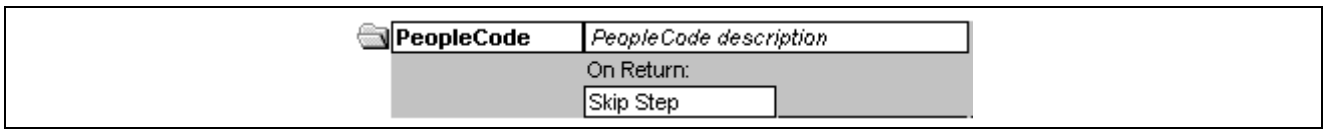
When accessing state records with PeopleCode, keep the following in mind:

- State records are unique to Application Engine programs.
- Within Application Engine PeopleCode, state record values can be accessed and modified using the standard `record.field` notation.

Note. When you launch an Application Engine program from PeopleSoft Process Scheduler, you can generate a process warning status on completion of the program by including and modifying the AE_APPSTATUS field in a state record. You can generate the warning status by setting AE_APPSTATUS to a value of 1.

Using If/Then Logic

From PeopleCode, you can trigger an error status, or false return, by using the Exit function. Use the On Return value on the PeopleCode action properties to specify how your Application Engine program behaves according to the return of your PeopleCode program. The following illustration shows the On Return property:



On Return action property

By default, the program terminates, similar to what happens when a SQL error occurs. But by changing the On Return value to *Skip Step*, you can control the flow of your Application Engine program.

You can use Exit to add an If condition to a step or a section break. For example,

```
If StateRec.Field1 = 'N'
Exit(1);
Else
/* Do processing */
End-if;
```

You must specify a non-zero return value to trigger the On Return action. The concepts of “return 1” and “return True” are equivalent. So, if the return value is non-zero or True, then PeopleSoft Application Engine performs what you specify for On Return, as in *Abort* or *Skip Step*. However, if the program returns zero, or False, PeopleSoft Application Engine ignores the selected On Return value.

Using PeopleCode in Loops

You can insert PeopleCode inside of a Do loop, but take care when using PeopleCode inside of high-volume Do loops (While, Select, Until). Keep the number of distinct programs inside the loop to a minimum. You should avoid having PeopleCode performing the actual work of the program and instead use it primarily to control the flow (If, Then logic), build dynamic SQL, or interact with external systems.

Using bind variables instead of literals to pass values to SQL statements is essential in PeopleCode loops or if the PeopleCode gets called in a loop. If the PeopleCode loops, there is a good probability that PeopleSoft Application Engine will use a dedicated cursor, which saves the overhead of recompiling the SQL for all iterations. If the PeopleCode gets called from within a loop, PeopleSoft Application Engine does not reduce the number of compiles, but Application Engine avoids flooding the SQL cache (for those database servers that support SQL cache) when it uses bind variables. Do not use bind variables for values in a Select list or for SQL identifiers, such as table and column names, as some databases do not support this.

Note. Null bind values of type DateTime, Date, or Time are always resolved into literals.

On those database platforms for which PeopleSoft has implemented this feature, Setting BulkMode to True often results in significant performance gains when inserting rows into a table in a loop.

In general, avoid PeopleCode calls within a loop. If you can call the PeopleCode outside of the loop, use that approach. This can increase overall performance.

Using the AESection Class

The AESection PeopleCode class enables you to change the properties of an Application Engine program section dynamically, without having to modify any of the PeopleSoft Application Engine tables directly. This enables you to develop rule-based applications that conform dynamically to variables that a user submits through a page, such as the Application Engine Request page.

The AESection class provides the following flexibility:

- Portions of SQL are determined by checks prior to execution.
- The logic flow conforms as rules change, and the program adjusts to the rules.

When using an AESection object, keep the following in mind:

- Check to make sure that you primarily require dynamic capabilities with the SQL your program generates.
- Make sure that the rules to which your program conform are relatively static, or at least defined well enough such that a standard template could easily accommodate them.
- Consider using SQL definitions to create dynamic SQL for your programs to avoid the complexity created by the AESection object using the StoreSQL function.
- The AESection class is designed to dynamically update only SQL-based actions, not PeopleCode, Call Section, or other actions.

  You can add a PeopleCode action to your generated section, but you can not alter the PeopleCode.
- The AESection class is designed for use for online processing.

Typically, dynamic sections should be constructed in response to a user action.

Note. Do not call an AESection object from an Application Engine PeopleCode action.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "AESection Class"

Making Synchronous Online Calls to Application Engine Programs

To make synchronous online calls to an Application Engine program, use the PeopleCode function CallAppEngine.

Note. If you make a synchronous call, the user can't perform another PeopleSoft task until the Application Engine program completes. Consider the size and performance of the Application Engine program called by CallAppEngine. You need to be sure that the program will run to successful completion consistently within an acceptable amount of time.

If an Application Engine program called by CallAppEngine terminates abnormally, the user receives an error, similar to other save time errors, that forces the user to cancel the operation. The CallAppEngine function returns a value based on the result of the Application Engine call. If the program was successful, it returns a zero, and if the program was unsuccessful, it returns a non-zero.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CallAppEngine

Using the File Class

The file layout class enables you to perform file input and /output operations with PeopleSoft Application Engine using PeopleCode. With a file object, you can open a file (for reading or writing), read data from a file, or write data to it. Using the combination of the file class and PeopleSoft Application Engine provides an effective method to integrate (or exchange) the data stored in a legacy system with your PeopleSoft system. The file class facilitates the creation of a flat file that both your legacy system and Application Engine programs support.

An Application Engine program running on the application server uses a file object to read the file sent from the legacy system and translate it, so that the file can update the affected PeopleSoft application tables. For the PeopleSoft system and the legacy system to interoperate, you need to first construct a file object that is compatible for both systems to insert and read data.

Attain rowset and record access for a file using a file layout definition. You create the file layout definition in PeopleSoft Application Designer, and it acts as a template for the file that both systems read from and write to. This simplifies reading, writing, and manipulating complex transaction data with PeopleCode.

Generally, use the file class and Application Engine combination when you can't implement the PeopleSoft Integration Broker solution.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "File Class"

Calling COBOL Modules

Using the PeopleCode RemoteCall function, you can call COBOL modules from a PeopleCode action. This option supports existing Application Engine programs that call COBOL modules. You can also use it to upgrade Application Engine programs from previous releases.

PTPECOBL Program

The PTPECOBL interface program is a PeopleSoft executable that enables you to invoke your called COBOL module and pass it required values. You code the RemoteCall function to invoke PTPECOBL, which in turn calls the specified COBOL module.

If you use PTPECOBL, you don't have to write your own executable to process this task. However, PTPECOBL does not perform any SQL processing other than retrieve a list of state record values. Consequently, if your current logic requires prior SQL processing, you may want to write your own executable file to call your COBOL module. In most situations, PTPECOBL saves you from having to write a custom executable file to handle each call to a generated dynamically loadable code (.GNT) file.

PTPECOBL performs the following tasks:

1. Initializes the specified state record in memory.
2. Invokes the COBOL module specified in your PeopleCode.
3. Submits required parameters to the called COBOL module.
4. Updates the state record as necessary, issues a commit, and then disconnects from the database after your program completes.

Note. While your COBOL program runs, it can access and return values to the state record.

Shared Values in PeopleSoft Application Engine and COBOL

Note the following options for sharing values between the Application Engine program and your called COBOL program:

- Use state records.

  If you add field names, PeopleSoft Application Engine enables you to pass state record values to the called COBOL program and to get changes passed back to the calling PeopleCode program. If you pass the state record values in this manner, use PTPECACH to retrieve and update values just as PTPEFCNV does.

- Code custom SQL.

  If you do not pass the initial values using state record fields, you need to insert the appropriate SQL in your called COBOL module to retrieve the appropriate values. Then, to return any updated values to the calling Application Engine program, you must insert the appropriate SQL into a PeopleCode program.

  If your COBOL program needs values that do not appear in a state record field, then you can pass PeopleCode variables and values. These variables and values are then retrieved and updated by calling PTPNETRT from within your COBOL program.

- Create a custom executable file.

  If you include extra SQL processing and use non-state record values, for consistency purposes, it might be a better approach to create a custom executable file. This way, you can call your program directly and have it perform all the PTPNETRT processing. Remember that a RemoteCall command can only call an executable program, not a GNT file.

Syntax and Parameters

The following example shows a sample RemoteCall function, issued from an Application Engine PeopleCode action to a COBOL module.

```
RemoteCall ("PSRCCBL",?
 "PSCOBOLPROG", "PTPECOBL",?
 "AECOBOLPROG", "MY_GNT",?
 "STATERECORD", "MY_AET",?
 "PRCSINST", MY_AET.PROCESS_INSTANCE,?
 "RETCODE", &RC,?
 "ERRMSG", &ERR_MSG,?
 "FIELD1", MY_AET.FIELD1,?
 "FIELD2", MY_AET.FIELD2);
```

Parameters	Description
PSRCCBL	This is the Remote Call dispatcher. It executes the specified COBOL program using the connect information of the current operator.
PSCOBOLPROG	Specify the name of the COBOL program to run. In this case, it is PTPECOBL. This parameter makes the remote call from PeopleSoft Application Engine distinct from a normal remote call. When you enter this parameter, you in effect enable the following parameters, some of which are required.
AECOBOLPROG	Specify the name of the COBOL module you're calling; for example, MY_GNT.
STATERECORD	Specify the appropriate state record that your Application Engine program will share with your COBOL module; for example, MY_AET. PTPECOBL then reserves space in memory for all of the fields on the state record, regardless of whether they will ultimately store values for processing.
PRCSINST	Specify the state record and Process Instance field; for example, MY_AET.PROCESS_INSTANCE. This retrieves the current process instance value that appears on the state record and submits it to your COBOL module using PTPECOBL.
RETCODE and ERRMSG	(Optional) Include RETCODE if you need to return information about any potential problems that the COBOL processing encountered, or use it if your Application Engine program must know whether it completed successfully.
Fieldnames and Values	This is where you specify any fields on the state record that contain initial values for your COBOL module. The quoted field names you specify must exist on the specified state record. The corresponding value can be a PeopleCode variable, a record.field reference, or a hardcoded value.

Commit and RemoteCall

Note the following using RemoteCall and an Application Engine program:

- The called COBOL module executes as a separate unit of work.
- Execute a commit in the step immediately preceding the step containing the RemoteCall PeopleCode action and also in the step containing the Remote Call PeopleCode action.

This enables the COBOL process to recognize the data changes made up to the point that it was called, and it also minimizes the time when the process might be in a non-restartable state.

- If you insert SQL processing into your COBOL module, commit updates are made by your module.

  PTPECOBL does not issue any commits.

- If the intent of your COBOL process is to update the value of a passed state record field, then the calling Application Engine PeopleCode is responsible for ensuring that the state record field has been modified, and the Application Engine program is responsible for committing the state record updates.

- Consider how your COBOL module will react in the event of a restart.

  Because the work in COBOL will have already been completed and committed, will your module ignore a duplicate call or be able to undo or redo the work multiple times? This is similar to issues faced when you execute a remote call from PeopleCode.

- Typically, when a COBOL program updates the database and then disconnects or terminates without having issuing an explicit commit or rollback, an implicit rollback occurs.

  Without an explicit commit, the database does not retain any updates.

Note. By default, RemoteCall doesn't generate any log files after the program completes. To generate and retain the .out and .err log files, you must set the *RCCBL Redirect* parameter in the PeopleSoft Process Scheduler configuration file to a value of *1*.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Process Scheduler, "Using the PSADMIN Utility," Editing the PeopleSoft Process Scheduler Configuration File

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," RemoteCall

Calling PeopleTools APIs

You can call all of the PeopleTools APIs from an Application Engine program. Keep the following items in mind when using APIs:

- All the PeopleTools APIs contain a Save method.

  However, when you call an API from your Application Engine program, regardless of the API's Save method, the data does not actually get saved until the Application Engine program issues a commit.

- If you've called a component interface from an Application Engine program, all the errors related to the API get logged in the PSMessage collection associated with the current session object.

- If you've sent a message, the errors get written to the message log and the Application Engine message log.

- If an Application Engine program called from message subscription PeopleCode encounters errors and the program exits (with `Exit (1)`), the error is written to the message log and is marked as an error

Using the CommitWork Function

This function commits pending changes (inserts, updates, and deletes) to the database. Keep the following in mind when using CommitWork:

- This function applies only to a batch Application Engine program.

If the program is invoked by CallAppEngine, the CommitWork function is ignored. The same is true for commit settings at the section or step level.

- This function can be used only in an Application Engine program that has restart disabled.
- The CommitWork function is useful only when you are doing row-at-a-time SQL processing in a single PeopleCode program, and you need to commit without exiting the program.

  In a typical Application Engine program, SQL commands are split between multiple Application Engine actions that fetch, insert, update, or delete application data. You use the section or step level commit settings to manage the commits.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CommitWork

Using PeopleCode Examples

The following sections provide some examples of common ways that you can utilize PeopleCode within Application Engine programs.

Do When Actions

Instead of a Do When action that checks a %BIND value, you can use PeopleCode to perform the equivalent operation. For example, suppose the following SQL exists in your program:

```
%SELECT(EXISTS) SELECT 'Y' FROM PS_INSTALLATION WHERE %BIND(TYPE) = 'X'),
```

Using PeopleCode, you could insert the following code:

```
If TYPE = 'X' Then
   Exit(0);
Else
   Exit(1);
End-if;
```

If you set the On Return parameter on the PeopleCode action properties to *Skip Step*, this code behaves the same as the Do When action. The advantage of using PeopleCode is that there is no trip to the database.

Dynamic SQL

If you have a Select statement that populates a text field with dynamic SQL, such as the following:

```
%SELECT(AE_WHERE1)
SELECT 'AND ACCOUNTING_DT  <= %Bind(ASOF_DATE)'
```

You can use this PeopleCode:

```
AE_WHERE1 = "AND ACCOUNTING_DT  <= %Bind(ASOF_DATE)";
```

Sequence Numbering

If you typically use Select statements to increment a sequence number inside of a Do Select, While, or Until loop, you can use the following PeopleCode instead:

```
SEQ_NBR = SEQ_NBR + 1;
```

Using PeopleCode rather than SQL is significant. Because the sequencing task occurs repeatedly inside a loop, the cost of using a SQL statement to increment the counter increases with the volume of transactions your program processes. When you are modifying a program to take advantage of PeopleCode, the areas of logic you should consider are those that start with steps that are executed inside a loop.

Note. You can also use the meta-SQL constructs %Next and %Previous when performing sequence numbering. Using these constructs may help performance in both PeopleCode and SQL calls.

Rowsets

You can use rowsets in Application Engine PeopleCode. However, using rowsets means you'll be using PeopleCode to handle more complicated processing, which degrades performance.

Math Functions

Use the math functions that your database offers whenever possible.

Internally, PeopleCode assigns types to numeric values. Calculations for the Decimal type are processed in arrays to ensure decimal point uniformity across hardware and operating system environments. This is much slower than calculations for type Integer, which are processed at the hardware level.

When PeopleCode converts strings to numeric values, it does so using the internal Decimal type. For performance reasons, avoid calculations using these values.

A third type of numeric value is the Float type. It is not used as frequently, for the following reasons:

- Constants are never stored as Float types in the compiled code.

  For example, 2.5 is always Decimal.

- The only way to produce a Float value is by using built-in functions, such as Float or the Financial math functions.

The Float type is used to produce a float result only if all operands are also of the Float type. Float operations occur at the hardware level.

PeopleCode does not offer optimum performance when processing non-Integer, non-Float math calculations. To perform calculations with these numeric types, consider allowing the database to perform the calculations in COBOL.

PeopleCode supports a range of mathematical functions and numeric types. Generally speaking, if a complex calculation is executed repetitively in an Application Engine program, careful analyis should be done to determine whether to perform the calculation in a PeopleCode action or to use the relational database management (RDBMS) functions through a SQL action. Using SQL may require PeopleSoft meta-SQL to handle platform differences, but it may be the most efficient way to update field values. If SQL is not appropriate, consider numeric typing in PeopleCode, as this affects the speed and accuracy of the calculation.

SQL Class

Instead of using the SQL class within PeopleCode, have Application Engine issue the SQL and use a Do Select action that loops around sections containing PeopleCode actions.

It might appear easier to code all of the logic within a single PeopleCode program, but splitting the logic into smaller pieces is preferable because you will have better performance, and you get a finer granularity of commit control. Within a PeopleCode program, you can commit in certain cases using the CommitWork function. You can always issue a commit between Application Engine steps.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," CommitWork.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "SQL Class," Understanding SQL Objects and PeopleSoft Application Engine Programs.

Arrays

Instead of using arrays in Application Engine PeopleCode, explore the use of temporary tables for storing pertinent or affected data. This has the following advantages:

- Data is available for restarts.
- An RDBMS is efficient at managing and searching tables.
- Using temporary tables also lends itself to set-based processing.

You can use the Statement Timings and PeopleCode Detail Timings trace options to generate an Application Engine timings report to determine whether your program is spending significant time processing arrays.

Including Dynamic SQL

Typically, developers include dynamic constructs in Application Engine programs to change SQL based on various runtime factors or on user-defined input entered through a page. There are a variety of ways to include dynamic SQL in Application Engine programs. For example, you could use:

- Dynamic sections, using the AESection object.
- Changing SQL, using the SQL class.
- References to SQL in your own tables.

The AESection class is primarily designed for online section building, and therefore won't be the most frequently used solution.

Use the SQL class to store SQL in a SQL definition that you can also access in PeopleSoft Application Designer. Then, if you have a few SQL statements to execute, generate the SQL IDs based on some methodology, such as a timestamp, and then store these in a table.

When the program runs, your SQL could query this table based on process and extract the appropriate SQL IDs to be executed with a SQL action in a Do Select loop.

```
%SQL(%BIND(MY_SQLID, NOQUOTES))
```

For a dynamic Do action, the AE_APPLID and the AE_SECTION fields must appear on the default state record.

Application Engine Meta-SQL Reference

This section describes the meta-SQL constructs, functions, and meta-variables you can use in PeopleSoft Application Engine.

Note. The SQL Editor does not validate all of the meta-SQL constructs, such as %Bind and %Select. Messages might appear stating these constructs are invalid.

%Abs

Description

Because the %Abs function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %Abs

%AeProgram

Description

Use the %AeProgram meta-variable to specify a quoted string containing the currently executing Application Engine program name.

%AeSection

Description

Use the %AeSection meta-variable to specify a quoted string containing the currently executing Application Engine section name.

%AeStep

Description

Use the %AeStep meta-variable to specify a quoted string containing the currently executing Application Engine Step name.

%AsOfDate

Description

Use the %AsOfDate meta-variable to specify a quoted string containing the as of date used for the current process.

%AsOfDateOvr

Description

Use the %AsOfDateOvr meta-variable only as a parameter of the %ExecuteEdits function, to override the default use of the system date with the value of a field on a joined record.

See Also

Chapter 5, "Using Meta-SQL and PeopleCode," %Table, page 89

%BINARYSORT

Description

Because the %BINARYSORT construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %BINARYSORT

%Bind

Syntax

%Bind([*recordname.*]*fieldname*[,**NOQUOTES**][,**NOWRAP**][,**STATIC**])

Description

Use the %Bind construct to retrieve a field value from a state record. Can be used anywhere in a SQL statement. When executed, %Bind returns the value of the state record field identified within its parentheses.

Notes About %Bind

Typically, when you use %Bind to provide a value for a field or a Where condition, the type of field in the state record that you reference with %Bind must match the field type of the corresponding database field used in the SQL statement.

On most platforms, you can't use a literal to populate a Long Varchar field. You should use the %Bind(*recordname.fieldname*) construct.

In the case of an external call to a section in another program, if the called program has its own default state record defined, then PeopleSoft Application Engine uses that default state record to resolve the %Bind(*fieldname*). Otherwise, the called program inherits the calling program's default state record.

All fields referenced by a %Select construct must be defined in the associated state record.

You must use the Date, Time, and DateTime output wrappers in the Select list that populates the state record fields.

This ensures compatibility across all supported database platforms. For example:

- First SQL action:

```
%Select(date_end)
   SELECT %DateOut(date_end )
      FROM PS_EXAMPLE
```

- Second SQL action:

```
INSERT INTO PS_EXAMPLE
   VALUES(%Bind(date_end))
```

Bind Variables and Date Wraps

The behavior of bind variables within PeopleSoft Application Engine PeopleCode and normal PeopleCode is the same.

Alternately, if you compare PeopleSoft Application Engine SQL to PeopleCode (of any type), then the system processes bind variables differently.

If you use the following approach:

```
AND TL_EMPL_DATA1.EFFDT <= %P(1))
```

Then in PeopleCode you issue:

```
%SQL(MY_SQL, %DateIn(:1))
```

This assumes that you have referenced the literal as a bind variable.

Or in PeopleSoft Application Engine SQL you issue:

```
%SQL(MY_SQL, %Bind(date_field))
%SQL(MY_SQL, %Bind(date_field, NOWRAP))
```

Parameters

Parameter	Description
Recordname	The name of a state record. If you do not specify a particular state record, PeopleSoft Application Engine uses the default state record to resolve the %Bind (*fieldname*).
Fieldname	The field defined in the state record.
NOQUOTES	If the field specified is a character field, its value is automatically enclosed in quotes unless you use the NOQUOTES parameter. Use NOQUOTES to include a dynamic table and field name reference, even an entire SQL statement or clause, in an Application Engine SQL action.

NOWRAP	If the field is of type Date, Time, or DateTime, the system automatically wraps its value in %DateIn or %DateOut, unless you use the NOWRAP parameter. Therefore, if the state record field is populated correctly, you don't need to be concerned with the inbound references, although you can suppress the inbound wrapping with the NOWRAP modifier inside the %Bind. Furthermore, PeopleSoft Application Engine skips the inbound wrapper if the %Bind (*date*) is in the select field list of another %Select statement. This is because the bind value is already in the outbound format, and the system selects it into another state record field in memory. In this circumstance there is no need for either an outbound wrapper or an inbound wrapper. For example, First SQL action:
	`%Select(date_end)`
	`SELECT %DateOut(date_end )`
	`FROM PS_GREG`
	Second SQL action:
	`INSERT INTO ps_greg`
	`VALUES(%Bind(date_end))`

STATIC	The STATIC parameter enables you to include a hard-coded value in a reused statement. For %Bind instances that contain dynamic SQL, this parameter must be used in conjunction with the NOQUOTES parameter for proper execution of a reused statement.

Example

```
UPDATE PS_REQ_HDR
   SET IN_PROCESS_FLG = %Bind(MY_AET.IN_PROCESS_FLG),
```

```
   PROCESS_INSTANCE = %Bind(PROCESS_INSTANCE)
WHERE IN_PROCESS_FLG = 'N'
   AND BUSINESS_UNIT || REQ_ID
   IN (SELECT BUSINESS_UNIT ||REQ_ID
      FROM PS_PO_REQRCON_WK1
      WHERE PROCESS_INSTANCE = %Bind(PROCESS_INSTANCE))
```

In the previous example, %Bind (PROCESS_INSTANCE) assigns the value of the field PROCESS_INSTANCE in the default state record to the PROCESS_INSTANCE field in table PS_REQ_HDR.

The %Bind construct is also used in a Where clause to identify rows in the table PS_PO_REQRCON_WK1, in which the value of PROCESS_INSTANCE equals the value of PROCESS_INSTANCE in the default state record.

%ClearCursor

Syntax

%ClearCursor({*program*, *section*, *step*, *action* | **ALL**})

Description

Use the %ClearCursor function to recompile a reused statement and reset any STATIC %Bind variables.

When you use the %ClearCursor function, keep the following in mind:

- The function must be located at the beginning of the statement.
- %ClearCursor can be the only function or command contained in the statement.

Parameters

Parameter	Description
program	Specify the name of the PeopleSoft Application Engine program containing the reused statement you want to recompile.
section	Specify the name of the section containing the reused statement you want to recompile.
step	Specify the name of the step containing the reused statement you want to recompile.
action	Specify one of the following values: • D: Do Select. • H: Do When. • N: Do Until. • W: Do While. • S: SQL.
ALL	Clear all cursors in the current PeopleSoft Application Engine program.

%COALESCE

Description

Because the %COALESCE function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %COALESCE

%Comma

Description

Use the %Comma meta-variable to specify a comma. This is useful where you must use a comma, but commas are not allowed due to the parsing rules. For example, you might use this if you wanted to pass a comma, as a parameter, to the %SQL meta-SQL function.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %SQL

%Concat

Description

Because the %Concat meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %Concat

%CurrentDateIn

Description

Because the %CurrentDateIn meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %CurrentDateIn

%CurrentDateOut

Description

Because the %CurrentDateOut meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %CurrentDateOut

%CurrentDateTimeIn

Description

Because the %CurrentDateTimeIn meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %CurrentDateTimeIn

%CurrentDateTimeOut

Description

Because the %CurrentDateTimeOut meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %CurrentDateTimeOut

%CurrentTimeIn

Description

Because the %CurrentTimeIn meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %CurrentTimeIn

%CurrentTimeOut

Description

Because the %CurrentTimeOut meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %CurrentTimeOut

%DateAdd

Description

Because the %DateAdd function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DateAdd

%DateDiff

Description

Because the %DateDiff function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DateDiff

%DateIn

Description

Because the %DateIn construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DateIn

%DateNull

Description

Because the %DateNull meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DateNull

%DateOut

Description

Because the %DateOut function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DateOut

%DatePart

Description

Because the %DatePart function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DatePart

%DateTimeDiff

Description

Because the %DateTimeDiff function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DateTimeDiff

%DateTimeIn

Description

Because the %DateTimeIn construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DateTimeIn

%DateTimeNull

Description

Because the %DateTimeNull meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DateTimeNull

%DateTimeOut

Description

Because the %DateTimeOut function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DateTimeOut

%DecDiv

Description

Because the %DecDiv function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DecDiv

%DecMult

Description

Because the %DecMult function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DecMult

%DTTM

Description

Because the %DTTM function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %DTTM

%EffDtCheck

Description

Because the %EffDtCheck construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %EffDtCheck

%Execute

Syntax

```
%Execute([/])
command1{; | /}
command2{; | /}...
commandN{; | /}
```

Description

Use the %Execute function to execute database-specific commands from within your Application Engine program. Also, the %Execute function enables you to include multiple statements in a single Application Engine action without encountering database-specific differences. For instance, there are instances where you could code a single Application Engine action to contain multiple SQL statements, and they may run successfully on one database platform. However, if you attempt to run the same code against a different database platform, you might encounter errors or skipped SQL.

By default, PeopleSoft Application Engine expects a semicolon to be used to delimit multiple commands within an %Execute function statement. You can instruct PeopleSoft Application Engine to use a forward slash (/) delimiter instead by placing a forward slash inside the function parentheses.

Note. When you use the %Execute function, it must be located at the beginning of the statement and can be the only function or command contained in the statement. The action type must be SQL.

Example

The following code enables you to use an Oracle PL/SQL block in an %Execute statement:

```
%Execute(/)
DECLARE
 counter INTEGER;
BEGIN
 FOR counter := 1 TO 10
  UPDATE pslock SET version = version + 1;
 END FOR;
END;
/
```

%ExecuteEdits

Syntax

%ExecuteEdits(*type*, *recordname* [*alias*][, *field1*, *field2*, ...])

Description

Use the %ExecuteEdits function to apply data dictionary edits in batch. The %ExecuteEdits function is Application-Engine-only meta-SQL. You can't use it in COBOL, SQR, or PeopleCode—not even in Application Engine PeopleCode.

Notes About %ExecuteEdits

Note the following:

- Consider performance carefully when using this function.

  Prompt table and Translate table edits have a significant impact, because they involve correlated subqueries. Run a SQL trace at execution time so that you can view the SQL generated by %ExecuteEdits. Look for opportunities where it can be optimized.

- In general, %ExecuteEdits is best used on a temporary table.

  If you must run this against a real application table, you should provide Where clause conditions to limit the number of rows to include only those that the program is currently processing. Process the rows in the current set all at once rather than processing them row by row.

- With %ExecuteEdits, you can't use work records in a batch, set-based operation.

  All higher-order key fields used by prompt table edits must exist on the record that your code intends to edit, and the field names must match exactly. For example,

  ```
  %ExecuteEdits(%Edit_PromptTable, MY_DATA_TMP)
  ```

  The record MY_DATA_TMP contains the field STATE with a prompt table edit against PS_REGION_VW, which has key fields COUNTRY and REGION. The REGION field corresponds to STATE, and COUNTRY is the higher-order key. For %ExecuteEdits to work correctly, the MY_DATA_TMP record must contain a field called COUNTRY. It's permissible for the edited field (STATE) to use a different name, because PeopleSoft Application Engine always references the last key field (ignoring EFFDT).

- In Application Engine, %ExecuteEdits uses the system date when performing comparisons with effective date (EFFDT); however, in some cases, this is not appropriate (Journal Edit, Journal Import, and so on). In these situations, Journal Date should be used when comparing with EFFDT. To override a program's use of the default system date with a selected field from a joined table, use %AsOfDateOvr. For example,

```
%ExecuteEdits(%AsOfDateOvr(alias.fieldname), %Bind(...)...)
```

- Restrict the number and type of edits to the minimum required.

  Don't perform edits on fields that are known to be valid, or that are given default values later in the process. Also, consider using a separate record with edits defined specifically for batch, or provide a list of fields to be edited.

Parameters

Parameter	Description
type	Specify any combination of the following (added together): • %Edit_Required • %Edit_YesNo • %Edit_DateRange • %Edit_PromptTable • %Edit_TranslateTable
recordname	Specify the record used to obtain the data dictionary edits.
field1, *field2*, ...	Specify a subset of the record's fields to which edits apply.

Example

Suppose you want to insert rows with missing or invalid values in three specific fields, selecting data from a temporary table but using edits defined on the original application table. Notice the use of an alias, or correlation name, inside the meta-SQL.

```
INSERT INTO PS_JRNL_LINE_ERROR (...)
SELECT ... FROM PS_JRNL_LINE_TMP A
WHERE A.PROCESS_INSTANCE = %BIND(PROCESS_INSTANCE)
  AND %EXECUTEEDITS(%Edit_Required + %Edit_PromptTable,?
 JRNL_LINE A, BUSINESS_UNIT, JOURNAL_ID, ACCOUNTING_DT)
```

To update rows in a temporary table that have some kind of edit error, you can use custom edits defined on the temporary table record:

```
UPDATE PS_PENDITEM_TAO
SELECT ERROR_FLAG = 'Y'
WHERE PROCESS_INSTANCE = %BIND(PROCESS_INSTANCE)
  AND %EXECUTEEDITS(%Edit_Required + %Edit_YesNo + %Edit_DateRange +?
 %Edit_PromptTable + %Edit_TranslateTable, PENDITEM_TAO)
```

%FirstRows

Description

Because the %FirstRows meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %FirstRows

%InsertSelect

Description

Because the %InsertSelect construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %InsertSelect

%JobInstance

Description

Use the %JobInstance meta-variable to specify the numeric (unquoted) PeopleSoft Process Scheduler job instance.

%Join

Description

Because the %Join construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %Join

%LeftParen

Description

Use the %LeftParen meta-variable to specify a left parenthesis. Usage is similar to %Comma.

See Also

Chapter 5, "Using Meta-SQL and PeopleCode," %Comma, page 71
Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %SQL

%Like

Description

Because the %Like construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %Like

%LikeExact

Description

Because the %LikeExact construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %LikeExact

%List

Syntax

```
%List({FIELD_LIST | FIELD_LIST_NOLONGS | KEY_FIELDS | ORDER_BY},
recordname [ correlation_id])
```

Description

The %List construct expands into a list of field names, delimited by commas. The fields included in the expanded list depends on the parameters.

Note. This meta-SQL is not implemented for COBOL, dynamic view SQL, or PeopleCode.

Considerations Using %List

When using %List in an Insert/Select or Insert/Values or %Select statement, you must have matching pairs of %List (or %ListBind) variables in the target and source field lists, using the same list type argument and record name to ensure consistency.

Parameters

Parameter	Description
FIELD_LIST	Use all field names in the given record. You can select only one option from FIELD_LIST, ORDER_BY, FIELD_LIST_NOLONGS, or KEY_FIELDS.
KEY_FIELDS	Use all key fields in the given record. You can select only one option from FIELD_LIST, FIELD_LIST_NOLONGS, KEY_FIELDS, or ORDER_BY.
ORDER_BY	Use all the key fields of *recordname*, adding the DESC field for descending key columns. This parameter is often used when the list being generated is for an Order By clause. You can select only one option from FIELD_LIST, KEY_FIELDS, ORDER_BY, or FIELD_LIST_NOLONGS.
FIELD_LIST_NOLONGS	Use all field names in the given record, except any long columns (long text or image fields.) You can select only one option from FIELD_LIST, ORDER_BY, KEY_FIELDS, or FIELD_LIST_NOLONGS.
recordname	Identify either a record or a subrecord that the field names are drawn from. This can be a bind variable, a record object, or a record name in the form *recname.* You cannot specify **RECORD.***recname*, a record name in quotation marks, or a table name.
correlation_id	Identify the single-letter correlation ID to relate the record specified by *recordname* and its fields.

Example

The following is a good example of using %List. Both the Insert and Select statements use the same %List variable.

```
INSERT INTO PS_PO_DISTRIB_STG (  %Sql(POCOMMONDISTSTGFLDLSTU)
,  %List(FIELD_LIST, CF16_AN_SBR)
,  MERCHANDISE_AMT
```

```
,  MERCH_AMT_BSE
,  QTY_DEMAND
,  QTY_PO
,  QTY_PO_STD
,  QTY_REQ)
SELECT  %Sql(POCOMMONDISTSTGFLDLSTU)
,  %List(FIELD_LIST, CF16_AN_SBR)
,  MERCHANDISE_AMT
,  MERCH_AMT_BSE
,  QTY_DEMAND
,  QTY_PO
,  QTY_PO_STD
,  QTY_REQ
FROM PS_PO_DIST_STG_WRK WRK
WHERE WRK.PROCESS_INSTANCE = %Bind(PROCESS_INSTANCE)
```

The following example shows a poor example of how to use %List. The Insert and Select field lists both use %List, but the Select field list is only partly dynamic. The rest is hard-coded.

```
INSERT INTO PS_EN_TRN_CMP_TMP (%List(FIELD_LIST, EN_TRN_CMP_TMP))
SELECT B.EIP_CTL_ID
, %List(SELECT_LIST, EN_BOM_COMPS A)
, E.COPY_DIRECTION
, E.BUSINESS_UNIT_TO
, E.BOM_TRANSFER_STAT
, 'N'
, B.MASS_MAINT_CODE
, 0
 FROM PS_EN_BOM_COMPS A
 , PS_EN_ASSY_TRN_TMP B
 , PS_EN_TRNS_TMP E
WHERE ...
```

The following example shows the previous poor example rewritten in a better way:

```
INSERT INTO PS_EN_TRN_CMP_TMP (EIP_CTL_ID,
,  %List(FIELD_LIST, EN_BOM_COMPS)
,  COPY_DIRECTION
,  BUSINESS_UNIT_TO
,  BOM_TRANSFER_STAT
,  EN_MMC_UPDATE_FLG
,  MASS_MAINT_CODE
,  EN_MMC_SEQ_FLG01
,  ...
, EN_MMC_SEQ_FLG20)
SELECT B.EIP_CTL_ID
  , %List(FIELD_LIST, EN_BOM_COMPS A)
, E.COPY_DIRECTION
, E.BUSINESS_UNIT_TO
, E.BOM_TRANSFER_STAT
, 'N'
, B.MASS_MAINT_CODE
, 0
```

```
, ...
, 0
 FROM PS_EN_BOM_COMPS A
 , PS_EN_ASSY_TRN_TMP B
 , PS_EN_TRNS_TMP E
WHERE ...
```

The following code segment is another poor example. Only the field list of the Insert statement is dynamically generated, and the Select statement is statically coded. If the table STL_NET_TBL is reordered, the Insert statement will be incorrect.

```
INSERT INTO PS_STL_NET_TBL (%List(FIELD_LIST, STL_NET_TBL ) )
SELECT :1
, :2
, :3
, :4
, :5
, :6
, :7
,:8
FROM PS_INSTALLATION
```

The following code shows the previous poor example rewritten in a better way:

```
INSERT INTO PS_STL_NET_TBL (%List(FIELD_LIST, STL_NET_TBL))
VALUES (%List(BIND_LIST, STL_NET_TBL MY_AET))
```

%ListBind

Syntax

%ListBind({**FIELD_LIST** | **FIELD_LIST_NOLONGS** | **KEY_FIELDS**}, *recordname* [*State_record_alias*])

Description

The %ListBind meta-SQL construct expands a field list as bind references for use in an Insert/Value statement.

Note. This meta-SQL is not implemented for COBOL, dynamic view SQL, or PeopleCode.

Considerations Using %ListBind

When using %ListBind in an insert/select or insert/values or %Select statement, you must have matching pairs of %List or %ListBind in the target and source field lists, using the same list type argument and record name to ensure consistency.

Parameters

Parameter	Description
FIELD_LIST	Use all field names in a record. You can select only one option from FIELD_LIST, FIELD_LIST_NOLONGS, or KEY_FIELDS.
FIELD_LIST_NOLONGS	Use all field names in a record, except any long columns (long text or image fields). You can select only one option from FIELD_LIST, FIELD_LIST_NOLONGS, or KEY_FIELDS.
KEY_FIELDS	Use all key field names in a record. You can select only one option from FIELD_LIST, FIELD_LIST_NOLONGS, or KEY_FIELDS.
recordname	Identify either a record or a subrecord that the field names are drawn from. This can be a bind variable, a record object, or a record name in the form *recname.* You cannot specify **RECORD.***recname*, a record name in quotation marks, or a table name.
State_record_alias	Specify the Application Engine state record buffer that contains the values (this could be different than the record used to derive the field list). If missing, the default state record is assumed.

Example

```
INSERT INTO PS_TARGET (FIELD1, FIELD2, %List(FIELD_LIST, CF_SUBREC), FIELDN)⇒
 VALUES (%Bind(MY_AET.FIELD1), %Bind(MY_AET.FIELD2), %ListBind(FIELD_LIST, ⇒
CF_SUBREC MY_AET), %Bind(MY_AET.FIELDN))
```

%ListEqual

Syntax

%ListEqual({**ALL** | **KEY** }, *Recordname* [*alias*], *RecordBuffer* [, *Separator*])

Description

The %ListEqual construct maps each field, possibly to an alias with a %Bind value, with a separator added before each equality. Each field is mapped as follows:

```
alias.X = %Bind(recbuffer.X)
```

This construct can be used in the Set clause of an Update statement or in a Where clause.

Note. This meta-SQL is not implemented for COBOL, dynamic view SQL, or PeopleCode.

Parameters

Parameter	Description
ALL | **KEY**	Specify if you want all fields or just key fields.
recordname	Identify either a record or a subrecord that the field names are drawn from. This can be a bind variable, a record object, or a record name in the form *recname.* You cannot specify **RECORD.***recname*, a record name in quotation marks, or a table name.
alias	(Optional) Specify an alias to precede each field name.
RecordBuffer	Specify the record buffer for the bind variables (this could be different than the record used to derive the field list).
Separator	If you want to specify a logical separator, specify either AND or OR with this parameter. If you don't specify a separator, no logical separator is used; the value of a comma is used instead.

Example

```
UPDATE PS_TEMP

SET %ListEqual(ALL, CF_SUBREC, MY_AET)
WHERE %ListEqual(KEYS, TEMP, MY_AET, AND)
```

%Next and %Previous

Description

Use the %Next and %Previous functions to return the value of the next or previous field in a numbered sequence. These functions are valid in any Application Engine SQL action, and should be used when performing sequence-numbering processing. Typically, you use them in place of a %Bind construct. These functions use the current value of the number field as a bind variable, and then increment (%Next) or decrement (%Previous) the value after the statement is executed successfully. A number field indicates the numeric field on the state record that you have initially set to a particular value (as in 1 to start).

If the statement is a Select and no rows are returned, the field value is not changed. The substitution rules are the same as for %Bind. For example, if the ReUse property is enabled, then the field is a true bind (':n' substituted). Otherwise, inline substitution occurs.

Example

You could use these functions in an Update statement within a Do Select action.

- Do Select action

```
%SELECT(field1, field2, ...) SELECT key1, key2, ... FROM PS_TABLE WHERE ...
ORDER BY key1, key2, ..."
```

- SQL

```
UPDATE PS_TABLE SET SEQ_NBR = %Next(seq_field) WHERE key1 = %Bind(field1)
AND key2 = %Bind(field2) ...
```

With a Do Select action, the increment/decrement occurs once per execution, not once for every fetch. So unless your Do Select action implements the Reselect property, the value is changed only on the first iteration of the loop. Alternatively, with the Reselect property or Do While and Do Until actions, every iteration re-executes the Select statement and then fetches one row. With these types of loops, the value changes on every iteration.

See Also

Chapter 5, "Using Meta-SQL and PeopleCode," %Bind, page 68

%NoUpperCase

Description

Because the %NoUpperCase construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %NoUppercase

%NumToChar

Description

Because the %NumToChar construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %NumToChar

%ProcessInstance

Description

Use the %ProcessInstance meta-variable to specify the numeric (unquoted) process instance.

%ReturnCode

Description

Use the %ReturnCode meta-variable to evaluate or specify the return code of the last Application Engine program step performed. If the operation fails, breaks, or generates an error, %ReturnCode is set to one of the following types of return codes:

- Database (SQL) call errors.
- PeopleCode function errors.
- GEN_ERROR, when produced by general runtime exceptions.
- AE_ABORT, when produced by application or runtime logic, including some memory-related errors.

If the application process is not terminated, %ReturnCode is reset to the default value of *0* for each subsequent successful operation.

%RightParen

Description

Use the %RightParen meta-variable to specify a right parenthesis. Usage is similar to %Comma.

See Also

Chapter 5, "Using Meta-SQL and PeopleCode," %Comma, page 71
Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %SQL

%Round

Description

Because the %Round function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %Round

%RoundCurrency

Syntax

%RoundCurrency(*expression*, [ALIAS.]*currency_field*)

Description

Use the %RoundCurrency function to return the value of an amount field rounded to the currency precision specified by the field's Currency Control Field property, as defined in the PeopleSoft Application Designer Record Field Properties dialog box. For this function to work, you must have the Multi-Currency option selected on the PeopleTools Options page.

See *Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration*, "Using PeopleTools Utilities," Using Administration Utilities.

This function is an enhanced version of the Application Engine &ROUND construct that appeared in previous releases, and is valid only in Application Engine SQL; it is not valid for SQLExecs or view text.

You can use this function in the Set clause of an Update statement or the Select list of an Insert/Select statement. The first parameter is an arbitrary expression of numeric values and columns from the source tables that computes the monetary amount to be rounded. The second parameter is the control currency field from a particular source table (the Update table, or a table in the From clause of an Insert/Selectstatement). This field identifies the corresponding currency value for the monetary amount.

Note. Remember that the as of date of the Application Engine program is used for obtaining the currency-rounding factor. The currency-rounding factor is determined by the value of DECIMAL_POSITIONS on the corresponding row in PS_CURRENCY_CD_TBL, which is an effective-dated table.

If multicurrency is not in effect, the result is rounded to the precision of the amount field (either 13.2 or 15.3 amount formats are possible).

Example

```
UPDATE PS_PENDING_DST
```

```
SET MONETARY_AMOUNT =
%RoundCurrency( FOREIGN_AMOUNT * CUR_EXCHNG_RT, CURRENCY_CD)
WHERE GROUP_BU = %Bind(GROUP_BU) AND GROUP_ID = %Bind(GROUP_ID)
```

%RunControl

Description

Use the %RunControl meta-variable to specify a quoted string containing the current run control identifier. The run control ID is available to your program when using %RunControl, regardless of whether there's a row in the AEREQUEST table.

%Select

Syntax

%Select(*statefield1*[, *statefield2*]...[, *statefieldN*])
Select *field1*[, *field2*]...[, *fieldN*]

The *statefields* must be valid fields on the state record (they may be fieldname or recordname.fieldname, as with %Bind), and *fields* must be either valid fields in the From tables or hard-coded values.

Description

Use the %Select construct to identify the state record fields to hold the values returned by the corresponding Select statement. The %Select construct is required at the beginning of all Select statements. For example, you need one in the flow control actions and one in the SQL actions that contain a Select statement.

You use the %Select construct to pass variables to the state record, and you use the %Bind construct to retrieve or reference the variables.

Example

Consider the following sample statement:

```
%SELECT(BUSINESS_UNIT,CUST_ID)
SELECT BUSINESS_UNIT, CUST_ID
FROM PS_CUST_DATA
   WHERE PROCESS_INSTANCE = %BIND(PROCESS_INSTANCE)
```

The following steps illustrate the execution of the previous statement:

1. Resolve bind variables.

   The string %Bind(PROCESS_INSTANCE) is replaced with the value of the state record field called PROCESS_INSTANCE.

2. Execute the SQL Select statement.

3. Perform a SQL Fetch statement.

   If a row is returned, the state record fields BUSINESS_UNIT and CUST_ID are updated with the results. If the Fetch statement does not return any rows, all fields in the %Select construct retain their prior values.

Note. All fields referenced by a %Select construct must be defined in the associated state record. Also, aggregate functions always return a row, so they always cause the state record to be updated. As such, for aggregate functions, there is no difference whether you use %SelectInit or %Select.

%SelectInit

Syntax

```
%SelectInit(statefield1[, statefield2]...[, statefieldN])
Select field1[, field2]...[, fieldN]
```

The *statefields* must be valid fields on the state record (they may be fieldname or recordname.fieldname, as with %Bind), and *fields* must be either valid fields in the From tables or hard-coded values.

Description

Use the %SelectInit construct to identify the state record fields to hold the values returned by the corresponding Select statement.

The %SelectInit construct is identical to the %Select construct, with the following exception: if the Select statement returns no rows, %SelectInit reinitializes the buffers. In the case of a %Select construct where no rows are returned, the state record fields retain their previous values.

Note. For aggregate functions, there is no difference whether you use %SelectInit or %Select.

%Space

Description

Use the %Space meta-variable to specify a single space. Usage is similar to %Comma.

See Also

Chapter 5, "Using Meta-SQL and PeopleCode," %Comma, page 71
Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %SQL

%SQL

Description

Use the %SQL construct to specify a SQL object, which replaces the %SQL construct in a statement. This enables commonly used SQL text to be shared among Application Engine and PeopleCode programs alike. In PeopleSoft Application Engine, you use %Bind to specify bind variables. In PeopleCode SQL, you can use

```
:record.field
```

or

```
:1
```

If you create SQL objects that you plan to share between PeopleSoft Application Engine and PeopleCode programs, the %SQL construct enables you to pass parameters for resolving bind variables without being concerned with the difference in the bind syntax that exists between PeopleSoft Application Engine and PeopleCode. However, the base SQL statement that uses %SQL to represent a shared object with binds needs to be tailored to PeopleSoft Application Engine or to PeopleCode.

When a SQL object specified has more than one version, the database type always takes precedence. That is:

- If one or more versions of a SQL definition are found for the database type of the current database connection, and if any of the versions have an effective date less than or equal to the current date, the most recent version is used.

- If no versions are found for the current database type, or if all of the versions have effective dates greater than the current date, the system looks for an effective version of the SQL definition under the database type "generic". If no version is found, an error occurs.

Example

For example, assume that your SQL is similar to the following:

```
UPDATE PS_TEMP_TBL SET ACTIVE = %BIND(MY_AET.ACTIVE)
WHERE PROCESS_INSTANCE = %ProcessInstance
```

That would not be valid if the SQL ran in PeopleCode. However, if you define your SQL as shown, you could use parameters in %SQL to insert the appropriate bind variable:

```
UPDATE PS_TEMP_TBL SET ACTIVE = %P(1)
WHERE PROCESS_INSTANCE = %ProcessInstance
```

From PeopleSoft Application Engine, the base SQL, or source statement, might look like the following:

```
%SQL(SQL_ID, %BIND(MY_AET.ACTIVE))
```

The PeopleCode SQL may appear as the following:

```
%SQL(SQL_ID, :MY_AET.ACTIVE)
```

Note. You can use %SQL only to reference SQL objects created directly in PeopleSoft Application Designer. For instance, you can not use %SQL to reference SQL that resides within a section in an application library. Common SQL should be stored as a proper SQL object.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, “Meta-SQL,” %SQL

%SQLRows

Description

Use the %SQLRows meta-variable to specify whether a SQL action returned any rows.

Can be used in any PeopleSoft Application Engine SQL statement, but the underlying value is affected only by SQL actions. It is not affected by Do When, Do Select, Do While, and Do Until actions. For Select statements, the value can only be 0 or 1: row not found or rows found, respectively. It does not reflect the actual number of rows that meet the Where criteria. To find the number of rows that meet the Where criteria, code a Select Count (*) statement.

%Substring

Description

Because the %Substring function can be used in more than just Application Engine programs, it’s documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, “Meta-SQL,” %Substring

%Table

Syntax

%Table(*recname*)

Description

Use the %Table construct to return the SQL table name for the record specified with *recname.*

This construct can be used to specify temporary tables for running parallel Application Engine processes across different subsets of data.

Example

For example, the following statement returns the record PS_ABSENCE_HIST:

```
%Table(ABSENCE_HIST)
```

If the record is a temporary table and the current process has a temporary table instance number specified, then %Table resolves to that instance of the temporary table PS_ ABSENCE_HIST*nn*, where *nn* is the instance number.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %Table

%Test

Description

Because the %Test construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %Test

%TextIn

Description

Because the %TextIn construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %TextIn

%TimeAdd

Description

Because the %TimeAdd construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %TimeAdd

%TimeIn

Description

Because the %TimeIn construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %TimeIn

%TimeNull

Description

Because the %TimeNull meta-variable can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %TimeNull

%TimeOut

Description

Because the %TimeOut construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %TimeOut

%TimePart

Description

Because the %TimePart function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %TimePart

%TrimSubstr

Description

Because the %TrimSubstr function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %TrimSubstr

%Truncate

Description

Because the %Truncate function can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %Truncate

%TruncateTable

Syntax

%TruncateTable(*table name*)

Description

Use the %TruncateTable construct to invoke a bulk delete command on a table. It's functionally identical to a Delete SQL statement with no Where clause, but it is faster on databases that support bulk deletes. If you're familiar with COBOL, this construct is an enhanced version of the COBOL meta-SQL construct with the same name.

Some database vendors have implemented bulk delete commands that decrease the time required to delete all the rows in a table by not logging rollback data in the transaction log. For the databases that support these commands, PeopleSoft Application Engine replaces %TruncateTable with Truncate Table SQL. For the other database types, %TruncateTable is replaced with Delete From SQL.

You should commit after the step that immediately precedes the step containing the %TruncateTable statement. In general, it's best to use this construct early in your Application Engine program as an initialization task. In addition, avoid using this meta-SQL when your Application Engine program is started from the PeopleCode CallAppEngine function.

Unlike the COBOL version, PeopleSoft Application Engine determines if a commit is possible prior to making the substitution. If a commit is possible, PeopleSoft Application Engine makes the substitution and then forces a checkpoint and commit after the successful execution of the delete.

If a commit is not possible, PeopleSoft Application Engine replaces the meta-SQL with a Delete From string. This ensures restart integrity when your program runs against a database where there is an implicit commit associated with Truncate Table or where rollback data is not logged.

For databases that either execute an implicit commit for %TruncateTable or require a commit before or after this meta-SQL, replace %TruncateTable with an unconditional delete in the following circumstances:

- A commit is not allowed, as in within an Application Engine program called from PeopleCode.
- The program issues a non-select SQL statement since the last commit occurred. In such a situation, data is likely to have changed.
- You are deferring commits in a Select/Fetch loop within a restartable program.

Note. To use a record name as the argument for %TruncateTable (instead of an explicit table name), you must include a %Table meta-SQL function to resolve the unspecified table name. For example, to specify the record PO_WEEK as the argument, use the following statement: `%TruncateTable(%Table(PO_WEEK))`.

See Also

Chapter 5, "Using Meta-SQL and PeopleCode," %Table, page 89

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %TruncateTable

%UpdateStats

Syntax

%UpdateStats(*record name* ,[HIGH/LOW])

For example,

%UpdateStats(PO_WRK1)

The default is LOW.

Description

Use the %UpdateStats construct to generate a platform-dependent SQL statement that updates the system catalog tables used by the database optimizer in choosing optimal query plans. Use this construct after your program has inserted large amounts of data into a temporary table that will be deleted before the end of the program run. This saves you from having to use dummy seed data for the temporary table and having to update statistics manually.

Notes About %UpdateStats

For databases that either execute an implicit commit for %UpdateStats or require a commit before or after this meta-SQL, PeopleSoft Application Engine skips %UpdateStats in the following circumstances:

- A commit is not allowed, as in within an Application Engine program called from PeopleCode.
- The program issues a non-select SQL statement since the last commit occurred.

  In such a situation, data is likely to have changed.
- You are deferring commits in a Select/Fetch loop in a restartable program.

  PeopleSoft Application Engine skips %UpdateStats even if the previous condition is false.

The following table shows how the %UpdateStats construct is resolved by the supported database systems.

Database Function	Behavior
MSS %UpdateStats	Specifying LOW produces the statement UPDATE STATISTICS *tablename* Specifying HIGH produces the statement UPDATE STATISTICS *tablename* WITH FULLSCAN
Sybase %UpdateStats	LOW and HIGH = UPDATE ALL STATISTICS *tablename*

Database Function	Behavior

Oracle %UpdateStats	PeopleSoft now uses DDL templates (in PSDDLMODEL) to determine SQL statements for %UpdateStats. Use DDLORA.DMS to change.
	Specifying LOW produces the statement
	`ANALYZE TABLE [tablename] ESTIMATE STATISTICS;`
	Specifying HIGH produces the statement
	`ANALYZE TABLE [tablename] COMPUTE STATISTICS;`

DB2 UNIX %UpdateStats	Specifying LOW produces the statement
	`RUNSTATS ON TABLE tablename AND INDEXES ALL`
	Specifying HIGH produces the statement
	`RUNSTATS ON TABLE tablename WITH DISTRIBUTION AND DETAILED INDEXES ALL`

DB2 390 %UpdateStats	Uses a DDL model template (in PSDDLMODEL) to format a control statement for the DB2 UDB for OS390 and z/OS Runstats utility. Refer to the PeopleTools Installation Guide and the Administration Guide for more details on using %UpdateStats with DB2 UDB for OS390 and z/OS.
	Specifying LOW produces the statement
	`RUNSTATS TABLESPACE [DBNAME].[TBSPCNAME] TABLE([DBNAME].[TABLE]) SAMPLE 25 [INDEXLIST] REPORT NO SHRLEVEL CHANGE UPDATE ACCESSPATH`
	Specifying HIGH produces the statement
	`RUNSTATS TABLESPACE [DBNAME].[TBSPCNAME] TABLE([DBNAME].[TABLE]) [INDEXLIST] REPORT NO SHRLEVEL CHANGE UPDATE ACCESSPATH`

Informix %UpdateStats	Specifying LOW produces the statement
	`UPDATE STATISTICS MEDIUM FOR TABLE tablename`
	Specifying HIGH produces the statement
	`UPDATE STATISTICS HIGH FOR TABLE tablename`

%UpdateStats Database Considerations

The following table lists potential issues that you might encounter when using %UpdateStats.

Database	Consideration
Microsoft SQL Server Sybase UDB	PeopleSoft forces a commit before and after the %UpdateStats statement. Therefore, the system skips this meta-SQL if a commit is not allowed. For instance, a commit is not allowed in the following situations: • The Application Engine program is not running in batch mode. • You have issued non-Select/Fetch SQL (in which the data is likely to change) since the last commit. • You are deferring commits in a Select/Fetch loop within a restartable program.
Oracle	Oracle has an implicit commit after the %UpdateStats statement executes. Same behavior as previous consideration.
DB2 UDB for OS/390 and z/OS	For DB2 UDB for OS/390 and z/OS, %UpdateStats requires IBM stored procedure DSNUTILS running in an authorized Work Load Manager Application Environment. It is also highly recommeded that individual tables intended to be a target of the %UpdateStats function are segregated to their own tablespaces. Refer to the following documents for more details on using %UpdateStats: PeopleTools Installation Guide for DB2 UDB for OS/390 and z/OS; PeopleTools Administration Guide for DB2 UDB for OS/390 and z/OS. **Note.** You can trace information messages from the Runstats command on DB2 for z/os executed as a result of issuing %UpdateStats. To enable this trace, select the SQL Informational Trace check box on the Configuration Manager – Trace page.
Informix IBM UDB	%UpdateStats locks the table being analyzed on UDB and Informix. Therefore, use this meta-SQL only on tables that are not likely to be concurrently accessed by other applications and users. You might use %UpdateStats to analyze PeopleSoft Application Engine dedicated temporary tables.
All	%UpdateStats consumes a large amount of time and database resources if run against very large tables. Therefore, analyze permanent data tables outside of application programs. Also, if temporary tables are likely to grow very large during a batch run, run the batch program only with %UpdateStats enabled to seed the statistics data or when the data composition changes dramatically.

Disabling %UpdateStats

You can disable %UpdateStats in the following ways:

- Include the following parameter on the command line when running an Application Engine program:

```
-DBFLAGS  1
```

- Change the Dbflags=0 parameter in the PeopleSoft Process Scheduler configuration file (or PSADMIN) to Dbflags=1.

Using %UpdateStats With COBOL

You can use the %UpdateStats construct from SQL embedded in COBOL programs. Use this syntax:

```
%UpdateStats(tablename)
```

When you issue this construct from PeopleTools, the parameter is *record name.*

%Upper

Description

Because the %Upper construct can be used in more than just Application Engine programs, it's documented in the *PeopleTools 8.45 PeopleCode Language Reference PeopleBook.*

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %Upper

CHAPTER 6

Managing Application Engine Programs

This chapter discusses how to:

- Run Application Engine programs.
- Debug Application Engine programs.
- Restart Application Engine programs.
- Cache the Application Engine server.
- Free locked temporary tables.

Running Application Engine Programs

This section provides an overview of program execution options and discusses how to:

- Create process definitions.
- List process definition parameters.
- Start programs with the Application Engine Process Request page.
- Use PeopleCode to invoke Application Engine programs.
- Use the command line to invoke Application Engine programs

Understanding Program Execution Options

You execute Application Engine programs in one of the following modes: batch using PeopleSoft Process Scheduler, online using a PeopleCode function, and manually using the command line. The following table lists some differences between online and batch programs:

Online Execution	Batch Execution
Started by the CallAppEngine function from PeopleCode.	Started through PeopleSoft Process Scheduler.
Program runs quickly, synchronously, and at random times.	Programs run for longer amounts of time, asynchronously, and at scheduled times.

Online Execution	Batch Execution
Potential for simultaneous execution.	Can be designed for parallel execution for performance.
Uses the online temporary table pool.	Uses the batch/dedicated temporary table pool.

Batch Programs Using PeopleSoft Process Scheduler

This is the most typical mode of execution. You invoke programs that run in this mode using PeopleSoft Process Scheduler or the Application Engine Process Request page. Batch mode is also referred to as asynchronous execution, meaning that it runs independently in the background. PeopleSoft Application Engine runs on any operating system that PeopleSoft supports as an application server. If your site uses an operating system that is not supported for PeopleSoft Application Engine, you must run Application Engine programs on the application server. (The only exception is OS/390 [z/OS]).

To run Application Engine programs on the batch server, you must install BEA Tuxedo. This applies to both UNIX and to Microsoft Windows NT batch servers. If you run your batch server on the same server machine as your application server, then the application server and the batch server can share one BEA Tuxedo installation. If your batch server is separate from your application server, then you must install BEA Tuxedo on your batch server.

The TOOLBINSRV parameter in the PeopleSoft Process Scheduler configuration file determines where PeopleSoft Process Scheduler invokes an Application Engine program. For high-volume batch environments, specify the PS_HOME\bin\server\winx86 directory that exists on the same machine where the Application Engine program runs.

Online Programs Using PeopleCode

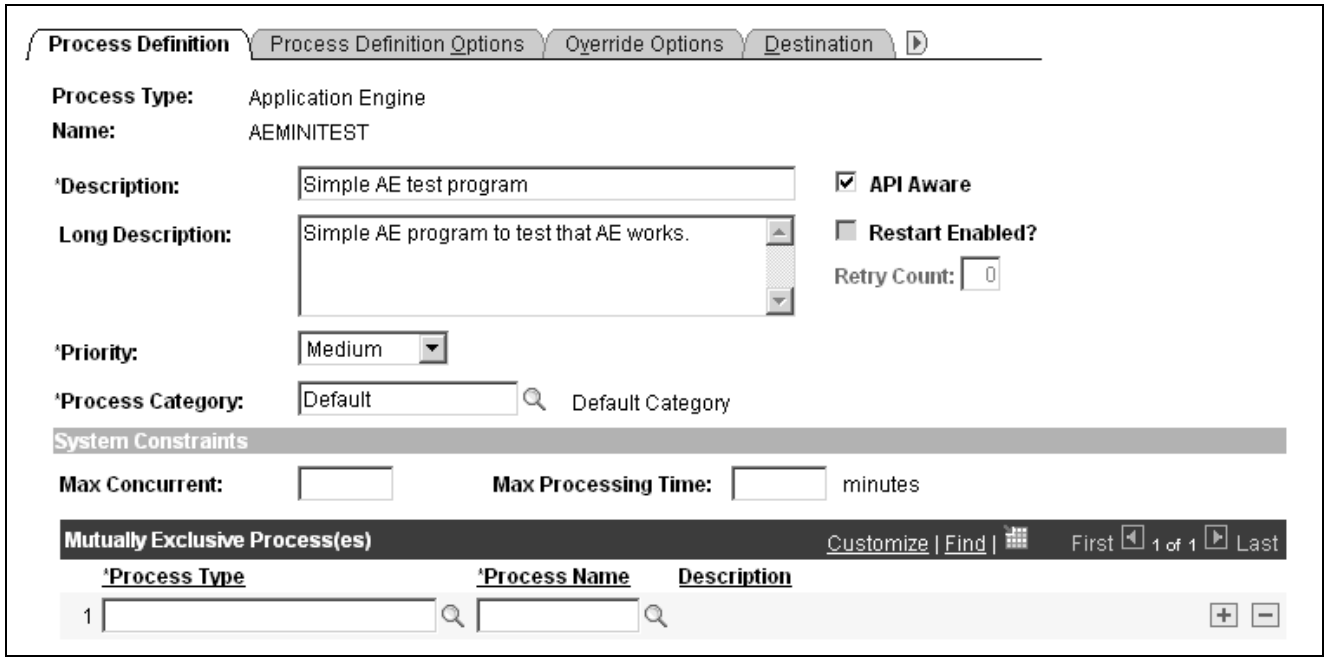
Application Engine programs that execute online are typically executed from a page with the CallAppEngine PeopleCode function. Such online processes are synchronous, meaning that subsequent processes wait on the results. For instance, a page may be frozen until the online process returns the necessary results. With the CallAppEngine function, there are no Commit statements issued. However, if you use the asynchronous online PeopleCode option, ProcessRequest, Commit statements are allowed.

Manual Programs Using the Command Line

Usually, you use this technique only during testing or if you need to manually restart a program.

Creating Process Definitions

Select PeopleTools, Process Scheduler, Processes to access the Processes - Process Definition page.



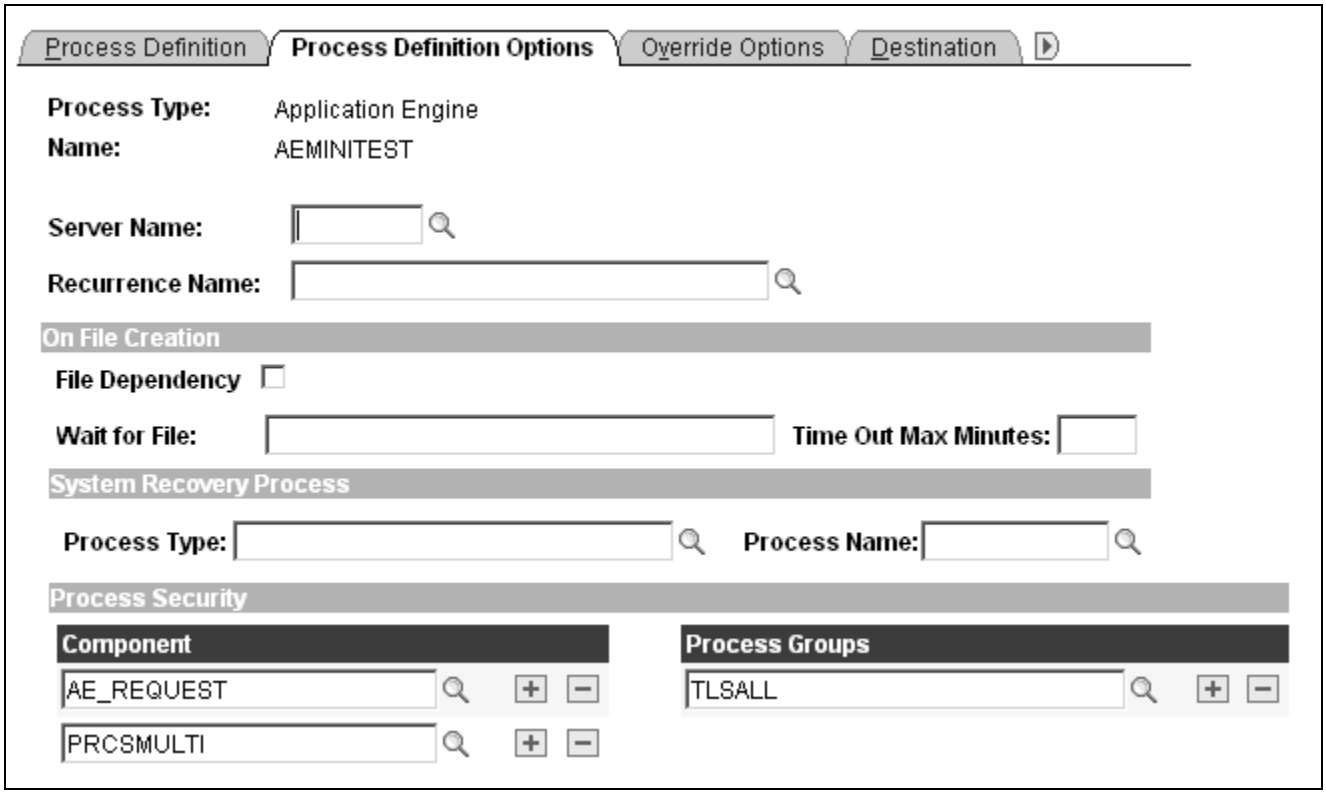
Processes - Process Definition page

To use PeopleSoft Process Scheduler for starting Application Engine batch programs, create a process definition for each program. Running Application Engine programs is very similar to running any COBOL or Structured Query Report (SQR) program that you typically invoke with PeopleSoft Process Scheduler. Use PeopleSoft Application Engine as the generic process type definition. Each Application Engine program that you invoke using PeopleSoft Process Scheduler requires a unique process definition derived from the generic process type definition.

Note. When creating a process definition based on the Application Engine process type definition, the process name you assign must exactly match your Application Engine program name.

Listing Process Definition Parameters

Access the Processes - Process Definition Options page.



Processes - Process Definition Options page

Use this page to list parameters. Here's the complete parameter list:

- -ct MICROSFT
- -cd %%DBNAME%%
- -co %%OPRID%%
- -cp %%OPRPSWD%%
- -r %%RUNCNTLID%%
- -i %%INSTANCE%%
- -ai %%PRCSNAME%

Starting Programs with the Application Engine Process Request Page

You can also start an Application Engine program by using the Application Engine Process Request page. Using this request page enables you to specify additional values and parameters than those that appear within PeopleSoft Process Scheduler process definitions.

Most users start Application Engine programs from an application-specific request page using PeopleSoft Process Scheduler. A systems expert or power user may, at times, need to create custom process requests that require multiple programs to perform parallel processing or that need to set specific, initial values in a state record. This is an example of where you might use the Application Engine process request page.

Note. Generally, if seed data or other PeopleSoft Application Engine request settings are required for a particular program, the application-specific request page has SQL executables that do the work transparent to the user. Typically, no user should invoke programs from the generic process request page. Use this page for internal testing and as a basis for designing program-specific request pages.

Tables Used in the Process Request Page

The Application Engine process request page inserts values into the following tables:

- AEREQUESTTBL.

  Contains all of the values that appear on the page except those in the Parameters group.

- AEREQUESTPARM.

  Includes only initial state record values specified in the Parameters group, if needed.

Note. Inserting a row in either of the Application Engine request tables is not required to run an Application Engine program. This is a key difference from versions of PeopleSoft Application Engine prior to PeopleTools 8, where a row in PeopleSoft Application Engine request tables is required to start a program regardless of how it is invoked. The run control ID is available to your program using %RunControl, whether or not there's a row inserted into the AEREQUESTTBL table.

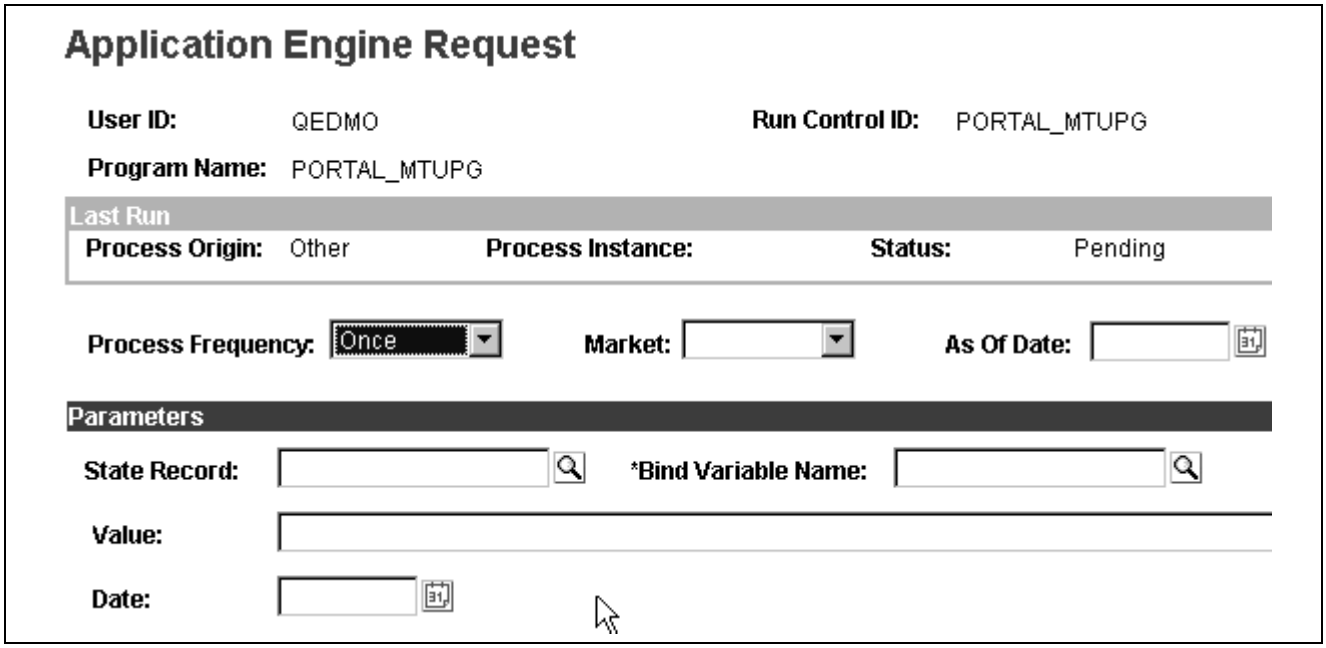
You need to use the Application Engine Request page to invoke Application Engine and insert a row into the Application Engine request records only if you need to perform any of the following tasks:

- Insert initial values into the state records associated with a particular program.
- Set an as of date for the Application Engine program to perform retroactive processing.
- Set a non-default market for the program.
- Set up a temporary table image to use if you are submitting a PeopleSoft EPM process request that performs parallel processing. Refer to PeopleSoft EPM application documentation for details.

Note. Entries in the AEREQUESTTBL table do not have any effect on Application Engine programs called from PeopleCode using the CallAppEngine function.

Application Engine Requests

Select PeopleTools, Application Engine, Request AE to access the Application Engine Request page.



Application Engine Request page

Process Origin Displays where the program was invoked: from PeopleSoft Process Scheduler, from the command line, and so on.

Process Instance Displays the process instance assigned to the previous program run.

Status Displays the status of the last program run, whether it is successful, pending, and so on.

Process Frequency Specify how long a particular process request will remain active or valid:

- *Always:* Select to run the process request as needed.
- *Once:* Select if a process request is a one-time-only request.
- *Don't:* Select to disable a process request, so that no one invokes it and potentially corrupts data.

As Of Date If you are requesting retroactive processing, specify the appropriate as of date.

Bind Variable Name Enter the appropriate field or bind variable for which you are inserting a value.

Value Enter the initial value that you want to set for the specified field.

Using PeopleCode to Invoke Application Engine Programs

To call a particular Application Engine program from a page using PeopleCode, use the CallAppEngine function in SavePreChange or SavePostChange PeopleCode. The basic syntax for CallAppEngine is as follows:

```
CallAppEngine(applid [, statereclist ]);
```

Note. The RemoteCall function is no longer valid for invoking Application Engine programs in PeopleCode. However, the RemoteCall function still applies to calling other COBOL functions. If you don't convert the RemoteCall PeopleCode that previously called an Application Engine program to use the new function, an error message appears.

Use CallAppEngine if the program you are invoking is a quick process. Because the process is synchronous, a user must wait for any process invoked by CallAppEngine to complete before doing anything else. If the called program causes an unreasonable delay, then use another alternative, such as the ScheduleProcess PeopleCode function.

Use CallAppEngine when you have a complex, SQL-intensive business process that must run in batch and online, or the process requires the use of dedicated temporary tables. If this is not the case, you are usually better off writing the entire program in native PeopleCode. If you've written logic in PeopleCode, presumably for online execution, and you want to reuse it in a batch program, you may be forced into row-by-row processing. Design the batch logic first, and then decide whether to have a separate online version or just reuse the batch code using CallAppEngine. Consider the trade-off between code reuse and performance. It is inherently more difficult, but not impossible, to develop a common solution that performs adequately in both batch and online environments.

Do not use CallAppEngine within an Application Engine PeopleCode step. If you need to call an Application Engine program from another Application Engine program, you must use the Call Section action.

Do not use CallAppEngine to control the commit operation. Programs called with CallAppEngine are embedded within a larger unit of work defined by the page trigger, such as a page save.

Note. Online PeopleCode that calls CallAppEngine should be set to run on the application server. You encounter performance issues if you run PeopleCode on the client in a three-tier configuration, because every SQL statement that PeopleSoft Application Engine issues must be serialized and then sent to the application server for execution.

Using the Command Line to Invoke Application Engine Programs

You might invoke an Application Engine program through the command line in the following situations:

- Restarting.

  When a program abends, a system administrator might restart the program using the command line. If needed, you can locate all of the specific program and process information from Process Monitor on the Process Request Detail dialog box. Normally, users (or system administrators) perform a restart from Process Monitor.

- Development or Testing.

  Many developers include the command line in a batch file to launch a program they are developing or testing. This way, they can quickly execute the batch file as needed. This also enables separation of development of the application program from its associated pages.

- Debugging.

  To debug a program running on the server, you can sign into the server (using telnet, for example) and invoke the program from the command line.

To start an Application Engine program from the command line, you must specify the Application Engine executable (PSAE.EXE) followed by the required parameters, as shown in the following example:

```
psae -CT dbtype -CS server -CD database_name -CO oprid -CP oprpswd?
 -R run_control_id -AI program_id -I process_instance -DEBUG (Y|N)?
 -DR (Y|N) -TRACE tracevalue -DBFLAGS flagsvalue -TOOLSTRACESQL value?
 -TOOLSTRACEPC value -OT outtype -OF outformat -FP filepath
```

Or, if your command line options are stored in a text file, you can enter:

```
psae optfilename
```

Note. For Microsoft Windows NT and UNIX servers, you must set the PS_SERVER_CFG environment variable before you invoke an Application Engine program from the command line. PS_SERVER_CFG must contain the fully qualified name of a correctly configured Process Scheduler PSPRCS.CFG file. When PeopleSoft Application Engine runs from the command line, it resolves %PS_SERVDIR% to the value of the environment variable PS_SERVDIR instead of the parent directory of a Process Scheduler configuration.

Command Line Options

-CT Specify the type of database to which you are connecting. Values are *MICROSFT*, *ORACLE*, *SYBASE*, *INFORMIX*, *DB2UNIX*, and *DB2ODBC*.

-CS Required for Sybase and Informix. For platforms that require a server name as part of sign-on, enter the appropriate server name. This affects Sybase, Informix, and Microsoft SQL Server. However, for Microsoft SQL Server, this option is valid but not required.

-CD Enter the name of the database to which the program will connect.

-CO Enter the user ID of the person who is running the program.

-CP Enter the password associated with the specified user ID.

-R Enter the run control ID to use for this run of the program.

-AI Specify the Application Engine program to run.

-I Required for restart. Enter the process instance for the program run. The default is 0, which means PeopleSoft Application Engine uses the next available process instance.

-DEBUG This parameter controls the Debug utility. Enter *Y* to indicate that you want the program to run in debugging mode, or enter *N* to indicate that you do not.

-DR This parameter controls restart disabling. Enter *Y* to disable restart, or enter *N* to enable restart.

-TRACE To enable tracing from the command line, enter this parameter and a specific trace value. The value you enter is the sum of the specific traces that you want to enable. Traces and values are:

1: Initiates the Application Engine step trace.

2: Initiates the Application Engine SQL trace.

128: Initiates the Application Engine timings file trace, which is similar to the COBOL timings trace.

256: Includes the PeopleCode detail timings in the 128 trace.

1024: Initiates the Application Engine timings table trace, which stores the results in database tables.

2048: Initiates the database optimizer explain, writing the results to the trace file. This option is supported only on Oracle, Informix, and Microsoft SQL Server.

4096: Initiates the database optimizer explain, storing the results in the Explain Plan table of the current database. This option is supported only on Oracle, DB2, and Microsoft SQL Server.

For example, to enable the 1, 2, and 128 traces, you would enter *131*,the sum of 1, 2, and 128. To indicate that you do not want any traces, enter *0*. If you don't explicitly enter *0*,PeopleSoft Application Engine uses the trace value set in PeopleSoft Configuration Manager.

-DBFLAGS To disable %UpdateStats meta-SQL construct, enter *1*.

-TOOLSTRACESQL Enable the SQL trace.

-TOOLSTRACEPC Enable the PeopleCode trace.

-OT (Optional) Initialize the PeopleCode meta-variable %OutDestType (numeric).

PeopleCode example of %OutDestType:

```
&ProcessRqst.OutDestType = %OutDestType ;
```

-OF (Optional) Initialize the PeopleCode meta-variable %OutDestFormat (numeric).

PeopleCode example of %OutDestFormat:

```
Query.RunToFile(Record QryPromptRecord, %OutDestFormat);
```

-FP (Optional) Initialize the PeopleCode meta-variable %FilePath (string).

PeopleCode example of %FilePath:

```
If All(%FilePath) Then
   &FILENAME = %FilePath | &FILENAME;
&MYFILE = GetFile(&FILENAME, "E", %FilePath_Absolute);
Else
  &MYFILE = GetFile(&FILENAME, "E", %FilePath_Relative);
End-If;
```

optfilename If you submit a file to PeopleSoft Application Engine as the first parameter in the command line, Application Engine reads the contents of the file and interprets the contents as if it were parameters entered on the command line. This option is intended mainly for the Microsoft Windows NT or UNIX Process Scheduler server environment. For example, you might enter *psae $temp/myparmfile.txt*

Note. For security reasons, after PeopleSoft Application Engine interprets the contents of the parameter file, it immediately deletes the file.

Debugging Application Engine Programs

This section discusses how to:

- Enable the PeopleSoft Application Engine debugger.
- Set debugging options.

Enabling the PeopleSoft Application Engine Debugger

To run a program in debug mode:

1. Set the debug option.

   You can set the debug option in the following locations:

   - Start PeopleSoft Configuration Manager and select the Process Scheduler tab.

     In the Application Engine group, enable debug by selecting the Debug check box. This is the method that applies to all methods of invocation.

   - If you used the command line option to invoke your Application Engine program, then you can include the −DEBUG Y parameter in the command line you submit to PSAE.EXE.

     If you already have the Debug check box selected in PeopleSoft Configuration Manager, then you do not need to include the −DEBUG parameter in your command line.

     Note. Setting debug capabilities in either PeopleSoft Configuration Manager or the command line turns debug mode on. However, if you have debug enabled in Configuration Manager and you submit −DEBUG N on the command line, the PeopleSoft Configuration Manager setting defines your default command line value, and the command line can override the default.

   - If you have PeopleCode in your Application Engine program, enable the PeopleCode debugger.

     When you launch your program and the PeopleCode action executes, you enter the PeopleCode debugger.

2. Execute the Application Engine program to debug.
3. At the Application Engine Debugger prompt, enter a command to enables a debugging option.

   Each command is represented by a single letter, such as X, L, or M. Enter the letter that corresponds to the option you want to engage. To see a list of the available debugging options, enter ? at the prompt.

To enable the PeopleCode debugger for Application Engine:

1. Sign on to PeopleTools using the same user ID that you are going to use to invoke the Application Engine program.
2. Open PeopleSoft Application Designer.
3. Select Debug, PeopleCode Debugger Mode.

   Your Application Engine program can be open on the desktop, but you do not need to open the Application Engine program or the PeopleCode action that you want to debug.
4. Select Debug, Break at Start.

   This causes the Application Engine program to break prior to executing any PeopleCode programs within it.

Setting Debugging Options

Each debugger option is represented by a single letter that you specify at the prompt. To engage the option you select, press ENTER.

Debugging Tips

Note the following tips about debugging programs:

- In some cases, such as when setting breakpoints or watch fields, submenus offer more options.

  After you are familiar with the commands, you can enter multiple items on the command line to combine commands and bypass the submenus. For example, to see a list of the breakpoints, you could enter B L.

  To set a field as a watch field, you could enter W S MY_FIELD.

Or, if it's on a different state record, enter W S MY_AET.MY_FIELD.

Note. The exception to this option is Modify, which always displays the current value and then prompts you to enter the new value. You can, however, enter M MY_AET.MY_FIELD to get directly to the new value prompt.

- The letter commands are not case-sensitive.

  For example, Q and q are valid commands.

Debugging Options

Option	Description
Quit	Enter Q. This option performs a rollback on the current unit of work in the debugging run, and it ends the debugging session. It effectively terminates your Application Engine program. Quit is useful for testing restart. Have some work committed and some uncommitted. Then, terminate the program at that point and roll back the pending work. You want to make sure the program restarts from the point of the last successful commit.
Exit	This option is valid only after one step has completed and another has not already begun. It is not valid once you reach the action level. Use this option as an alternative to Quit. Exit ends the program run and the debugging session, but it also commits the current unit of that the program has already completed. This option can be helpful when testing your restart logic.
Commit	Enter C. To commit the current unit of work in your program, use this option. It is valid only after a step has completed and before another has already begun. It is not valid once you reach the action level. You can use this option, for example, to use your database query tool to check the data in your database.
Break	Enter B. Sets a breakpoint. When the program reaches the breakpoint, it temporarily halts execution to enable you to observe the state of the current process. Breakpoint options include: Set: Enter S to set a breakpoint location. The breakpoint location defaults to the current location in the program, but you can specify other sections or steps by overriding the default values that appear in the brackets. Unset: Enter U to remove breakpoints previously set. List: Enter L to list breakpoints. When you enter this command, make sure that you have entered B first to specify the break option. If you just enter L from the main command prompt, you engage the Look option.

Option	Description
Look	Enter L. Enables you to observe the values currently in the state record associated with the program you are debugging. You must specify the state record at the Record Name prompt. By default, the default state record as specified in your program properties appears with the brackets. You can also specify a specific field name on the state record in the Field Name prompt. To look at all the fields in the state record, leave the asterisk (*) within the brackets unchanged.
Modify	Enter M. Enables you to modify the value of a state record value for debugging purposes. Suppose the previous steps did not set a value correctly. However, you may want to see how the rest of the program would perform if the appropriate value existed in the state record. This enables you to give your program some help in the debugging or testing phase. As with the Look command, you must specify the appropriate state record (if you are using multiple state records), and you must specify one field. You can modify only one field at time.
Watch	Enter W. When you specify a field as a watch field, the program stops when the value of the field changes. Similar to the Break command, you can specify options for Set, Unset, and List.
Step Over	Enter S. Executes the current step to completion and stop at the next step in the current section. The behavior depends on the current level or the program. You start at the step level, and then can step into the action level. If you are at the step level and use step over, you go to the next step in the current section, skipping over all actions (including any call sections). If you are at the action level, step over executes the current action and stops at the next action in the current step, or at the next step in the current section.
Step Into	Enter I. Use this option to observe a step or called section in a more granular level. For instance, you can check each SQL statement and stop. By using this option and checking the state record at each stop, you can easily isolate problem SQL or PeopleCode. As with Step Over, the behavior depends on the level. At the step level, you can step into the action level and stop before the first action in the step. At the action level, if the current action is a call section, Step Into takes you to the first step in the called section. For other action types, Step Into acts the same as Step Over, because there is no deeper level in which to step.

Option	Description
Step Out of	Enter O. After you've stepped into a step or called section, use the Step Out of option to run the rest of the current step or called section and stop. As with the previous step options, the behavior of Step Out of depends on the current level of the program. At the step level, Step Out of completes the remaining steps in the current section, returns to the calling section or step, and stops at the next action in that step. If the section is MAIN and is not called by another section or step, then Step Out of behaves the same as the Go option. At the action level, Step Out of completes the current step and stops at the next step in the current section, or if the program is at the end of a section, Step Out of returns to the calling section or step.
Go	Enter G. After the program has stopped at a specific location, and you've examined its current state, you can use the Go command to resume the execution of the program. This is a helpful command when you have breakpoints set. With this command, the program won't stop at a step or action; it only stops at the next breakpoint or watch field, or when the program runs to completion.
Run to commit	Enter R. Resumes execution of your program after it has stopped. This command forces the program to stop again after the next commit. This is a good option to use when observing your commit strategy and how it will affect a restart.

Example of the Look Option

To view the value stored in a specific field of the state record after a step or action, enter the appropriate field name at the Field Name prompt. For example, if you entered AE_TESTAPPL_AET at the Record Name prompt and AE_INT_6 at the Field Name prompt, you would see the value of the AE_INT_6 field in the AE_TESTAPPL_AET record.

You can also use an asterisk (*) as a wildcard to get a partial list. For example, if you enter AE_INT* at the Field Name prompt, you see only the fields that start with AE_INT. This is also true for the Record Name prompt. This is useful both to list multiple fields across multiple records or as a shortcut. If you know there is only one state record that starts with *XXX*, you don't have to type the full name—just enter *XXX*.

Example of the Modify Option

If you wanted to set the AE_INT_15 field in the AETESTPROG to 10, you would enter the record (AE_TESTAPPL_AET) at the Record Name prompt and the field (AE_INT_15) at the Field Name prompt.

You then see the currrent value of the field. At the prompt, you can enter a new value.

Using the Look command, you can check to see that the value you specified now exists in the state record.

Example of the Watch Option

Enter S to set a watch field. After you enter S, you enter the record name (such as AE_TESTAPPL_AET) and field name (such as AE_INT_7) at the appropriate prompts.

Enter U to unset, or remove, a watch field from the list. After you enter U, you see a list of active watch fields. You enter the watch field ID number to remove a field. For example, if the field AE_INT_7 were second in the watch field list, you would enter 2 to remove it.

After the completion of a step or action, enter L to list, or view, the values of all the fields that you have included in the watch list.

Note. You cannot set a watch on a long text field.

Restarting Application Engine Programs

One key feature of PeopleSoft Application Engine is its built-in checkpoint and restart capabilities. If there is an abnormal termination or failure on a step in the program, you can restart the request from the last successful checkpoint, or the step immediately preceding the step that failed. You restart the program from the process request page.

This section provides an overview of restart and discusses how to:

- Determine when to use restart.
- Control abnormal terminations.
- Restart Application Engine programs.
- Start Application Engine programs from the beginning.
- Enable and disable restart.

Understanding Restart

Application Engine programs save to the database (perform a commit) only when an entire program successfully completes. You must set any individual commits where appropriate.

At the section level, you can set a commit after each step in that section. At the step level, you can require or defer commits for individual steps, or you can increase the commit frequency within a step to *N* iterations of a looping action within a step, such as a Do Select or Do While action.

The commit level that you select plays a major role in how restart works in a program. Each time that PeopleSoft Application Engine issues a commit with restart enabled, it records the current state of the program. The recording of the current state that PeopleSoft Application Engine performs is referred to as a checkpoint.

Using the restart feature enables you to perform commits more often in a program. Restart reduces the overall impact on other users and processes while the background program is running, because it reduces the amount of rows that are locked by the program, allowing multiple instances of the program to run concurrently (parallel processing), which may be useful for high-volume solutions.

With restart, if a failure occurs at any point in the process, the user can restart the program and expect the program to behave in the following manner:

- Ignore the steps that have already completed up to the last successful commit.
- Begin processing at the next step after the last successful commit.

The ability for PeopleSoft Application Engine to“remember completed steps depends on a record called AERUNCONTROL, which is keyed by process instance.

When a program runs, each time PeopleSoft Application Engine issues a commit it also saves all of the information required for a program restart in the AERUNCONTROL record.

Determining When to Use Restart

Usually, you want to develop programs to take advantage of PeopleSoft Application Engine restart capabilities. Programs that are good candidates for restart do a lot of preparation work up front, like joining tables and loading data into temporary work tables. Also, programs that might put data in an unstable state if they terminate abnormally during a run should be considered to take advantage of restart. As a general rule, restart is essential for programs that primarily do set-based processing.

However, if your program has one the following characteristics, you may want to disable restart:

- It's mainly row-by-row processing.
- The overhead involved with PeopleSoft Application Engine performing a checkpoint during the program run is not desirable.
- The program commits after *N* iterations of a looping construct within a step, and the Select statement driving the loop is composed in such a way that if the program terminated and then started again, it would ignore transactions that were already processed in the previous program run. In this sense, the program processes the restart internally, in that PeopleSoft Application Engine treats each start of a program as a fresh start, instead of restarting a previous instance.

When developing for restart, consider the consequences if a program fails and you can't restart the program. Given the commit structure that you've defined for your Application Engine program, would your data remain in an usual state if a failure were to occur after any of the commits? Would it be easy to recover from such a case?

Using Restart at the Program Level

PeopleSoft Application Engine automatically performs all state record updates. When an Application Engine program starts, it inserts a row in the state record for the assigned process instance. Then the system updates the state record whenever the program performs a commit to store changed values into the database. Finally, the state record row is deleted upon successful completion of the application.

However, if the state record the program uses is a work record, no database updates can be made to the record. Consequently, if you restart the program, you might get unexpected results, because the memory was lost when the program terminated. In fact, the system reinitializes any state records that are work records at each commit, to ensure consistent behavior during a normal run and a restarted run. Therefore, you may need to make at least one of your state records a SQL table to contain values that must be retained across commits or in case of termination.

Finally, the other consideration for programming for restart at the program level is to check both the PeopleSoft Application Engine Program Properties dialog box and PeopleSoft Configuration Manager to make sure that Disable Restart check box is not selected.

Using Restart at the Section Level

The section level property associated with restart is section type, which has the options Prepare Only and Critical Updates.

If a section is only preparing data, as in selecting it, populating temporary tables, or updating temporary tables, then set the section type to Prepare Only. However, if the section updates permanent application tables in the database, set the option to Critical Updates.

During runtime, when the system arrives at the first section set to Critical Updates, it sets the AE_CRITICAL_PHASE value in the AERUNCONTROL record to *Y*. Once set, the value of AE_CRITICAL_PHASE remains *Y* until the program completes successfully. When the program completes, the corresponding row in AERUNCONTROL is deleted. Therefore, a Prepare Only section following the Critical Updates section won't reset the AE_CRITICAL_PHASE value to *N*.

If your program terminates, the user can check the AE_CRITICAL_PHASE value. If it's *Y.* the user knows that the section that failed is critical and that the program should be restarted to ensure data integrity. If AE_CRITICAL_PHASE is *N*, restarting may not be necessary; however, as a general rule, you should restart even if AE_CRITICAL_PHASE is set to *N.*

Using Restart at the Step Level

In your program's Where clause of a Do Select action, you should include conditions that reduce the answer set returned from the Select statement.

For example,

```
SELECT RECNAME, FIELDNAME
   FROM PS_AE_RECFIELD
   ORDER BY RECNAME, FIELDNAME
```

If you ran this Select statement as part of a Do Select action with *Restartable* selected as the Do Select type, the system might process some of the rows twice after a restart. Also, if you have specified *Reselect*, the program could execute in an infinite loop, because there's nothing to reduce the answer set. However, if you modified the Select statement to look more like the following, you could make it *Restartable.*

```
SELECT RECNAME, FIELDNAME
FROM PS_AE_RECFIELD
WHERE RECNAME > %Bind(RECNAME)
OR (RECNAME = %Bind(RECNAME) AND FIELDNAME > %Bind(FIELDNAME))
ORDER BY RECNAME, FIELDNAME
```

A Do Select action that has been coded for *Restartable* can be converted to *Select/Fetch*, but the opposite is not true.

The previous example shows the use of a key column to reduce the answer set. This can be convenient if your record has only one or two key fields. However, if your record has three or four keys, your SQL would become overly complex.

Instead of matching key fields, you could add a switch to the selected table, and then have the processing of the called section modify the switch as it processes the row. In this example, your Select statement could look like the following:

```
SELECT COLUMN1, COLUMN2, . . .
   FROM PS_TABLE1
   WHERE PROCESSING_SWITCH='N'. . .
```

Controlling Abnormal Terminations

A controlled abnormal termination (sometimes called an *abend)* means that Application Engine exits gracefully because of a calculated error condition. Some examples of controlled abends are:

- SQL errors while you have set On Error to*Abort.*
- A PeopleCode return value, when On Return is set to *Abort.*
- A SQL statement that affects no rows, when you have set On No Rows to *Abort.*

In these situations (when PeopleSoft Application Engine is in control) the Run Status field in Process Monitor reads *Error.*

An uncontrolled termination occurs when there is a memory violation or a user terminates a process. In these cases, the Run Status field in Process Monitor shows *Processing.*

Restarting Application Engine Programs

There are two ways to restart an Application Engine program:

- From the command line.
- From a process request page.

Note. The following procedures for restarting a failed Application Engine program assume that you have rectified the error that caused the program to fail in the first place. For instance, suppose the name of a referenced table has changed. Regardless of how many times you restart the program, it will continue to fail until you've modified references to the old table name.

Restarting from the Command Line

Normally, only developers and system administrators use the command line for restarting Application Engine programs. Users, in most cases, should not be expected to use this method.

You can use the command line option to restart programs that run on the client or the server. PeopleSoft Application Engine references only the process instance of the failed process. Therefore, if you run a process on the client and it fails, you can restart it from the server using the server command line. Likewise, if you run a process from the server and it fails, you could restart it from the client using the command line.

To restart an Application Engine program from the command line:

1. Collect the command line values associated with the failed program.

   These values include database type, database name, user ID and password, run control ID, program name, and the process instance. You can find these variables on the Process Details dialog box, the corresponding state record, or the Application Engine Run Control table. Where the values reside depends on how you invoked the program. For instance, if you invoked the program using the command line, or outside of PeopleSoft Process Scheduler, then you cannot view details associated with the program run on the Process Details dialog box.

2. Enter the following command line syntax at the command prompt substituting the values from the previous step.

   ```
   PSAE.EXE –CT DB_TYPE -CD DB_NAME -CO OPRID -CP PASSWORD⇒
    -R RUN_CONTROL -AI PROGRAM_NAME -I PROCESS_INSTANCE
   ```

   Note. Some database platforms, such as Sybase, also require that you include a server name in the argument list.

Restarting from the Process Request Page

You can restart programs from a process request page only for those programs that run on the server.

To restart an Application Engine program from a process requests page:

1. Open PeopleSoft Process Scheduler by selecting PeopleTools, Process Scheduler, System Process Requests.
2. Locate the run control ID number of the program to restart.
3. To display the details of the failed process, click the Process Detail link.
4. On the Process Request Details page, select Restart Request, and click OK.

Bad Restart Error

If you attempt to restart what PeopleSoft Application Engine believes to be a process that completed successfully, you receive a bad restart message. You can also get this message if your Application Engine application is defined with restart disabled.

Starting Application Engine Programs from the Beginning

When an Application Engine program ends abnormally, you may have to decide whether you should restart the process or just start it from the beginning. As your Application Engine program ran at least part way through, starting over may leave your data in an unknown state. Also, application logic might need to be undone, depending on what stage the program was at when it failed, what data the program had committed, and so on.

However, if restart is enabled and you attempt to start a new process that matches the run control ID and user ID for another process, you receive a suspend error. Because the process instance for these two processes is different, the new request fails. This usually occurs when a user tries to run the program again after receiving an error on the previous attempt.

To start the program over from the beginning, you can use SQL to delete the row that corresponds to the failed program from the Application Engine run control table and your state record.

To restart an Application Engine program from the beginning:

1. Open your native SQL editor and manually delete the row in PS_AERUNCONTROL that corresponds to the program you want to start from the beginning.

   Use the following SQL to accomplish this step:

   ```
   DELETE FROM PS_AERUNCONTROL
   WHERE OPRID=OPRID
   AND RUN_CNTL_ID=Run_Control_ID
   ```

2. Delete from your state record the row that corresponds to the failed program run.

   Use the following SQL to accomplish this step:

   ```
   DELETE FROM PS_MY_AET
   WHERE PROCESS_INSTANCE=Process_Instance
   ```

Note. To restart the program, you can also select Restart Request from the Process Request Details dialog box.

Enabling and Disabling Restart

To disable restart, use any of these methods:

- Select the Disable Restart check box on the PeopleSoft Application Engine Program Properties dialog box.

  To access program properties, select File, Definition properties, and select the Advanced tab.

- Select the Disable Restart check box in the Configuration Manager profile.

  To access the profile, start Configuration Manager, select the Profile tab, and click Edit. Then select the Process Scheduler tab.

- Include the −DR Y option in the command line of PSAE.EXE.

If you've disabled restart in any of these three places, restart is disabled.

Therefore, if you want the program to restart in a production environment, while still keeping a restart condition from getting in the way during development and testing, you can clear the Disable Restart check box in the Application Engine program properties. Then during development you can select the Disable Restart check box in Configuration Manager, or invoke your program from the command line with the -DR Y option, without having to reconfigure the program for testing.

Caching the Application Engine Server

PeopleSoft Application Engine caches metadata, just like the application server. This enhances performance, because a program can refer to the local cache for any objects that it uses.

Cache Directory Location

Application Engine programs that run on a Microsoft Windows NT or UNIX server each lock their own cache directory for the duration of the run. These directories are found under the master cache directory. The master directory is created under the directory specified by the CacheBaseDir variable in the PeopleSoft Process Scheduler configuration file. If all existing cache directories are locked, a new one is created. Cache subdirectories are named sequentially, starting at 1.

If you do not enter a fully qualified path for the CacheBaseDir variable, then PeopleSoft Application Engine creates the cache directory within the directory in which the program is set to run.

Note. Do not share the CacheBaseDir variable with application servers, and do not use environment variables when specifying CacheBaseDir, because the system does not resolve them. For example, do not set CacheBaseDir to $PS_HOME.

Cache Parameters

In the PSPRCS.CFG (PS_SERVER_CFG) file, there are two additional cache parameters. They are:

- Enable Server Caching
- Server Cache Mode

Do not alter these settings from the delivered defaults. These settings are reserved for future use.

Freeing Locked Temporary Tables

If you use dedicated temporary tables for Application Engine programs, then you might need to free, or unlock, a temporary table if the program running against it terminates abnormally. Because most Application Engine programs run through PeopleSoft Process Scheduler, typically you use Process Monitor to unlock the temporary tables. Deleting or restarting a process using Process Monitor automatically frees the locked temporary tables.

For the programs that you invoke outside of PeopleSoft Process Scheduler, use the Manage Abends page. Programs running outside of Process Scheduler include those invoked from CallAppEngine PeopleCode and the command line.

To free locked temporary tables using the Manage Abends page:

1. Select PeopleTools, Application Engine, Manage Abends.

2. Identify the program that has the particular temporary tables locked.

   You can uniquely identify programs using the process instance, run control ID, program name, user ID, and run date and time columns.

3. Click the Temp Tables link.

4. On the Temporary Tables page, click the Release button to unlock the temporary tables associated with the program.

CHAPTER 7

Calling Application Engine Programs from COBOL

To facilitate the conversion of existing COBOL programs to Application Engine programs, you can call Application Engine programs from existing COBOL code.

This chapter discusses how to:

- Add copybooks to COBOL programs.
- Assign copybook values.
- Handle COBOL errors.

Adding Copybooks to COBOL Programs

To enable you to call Application Engine programs from COBOL programs, include the copybook called PTCCBLAE.CBL with your COBOL programs. This copybook is located in *PS_HOME*\src\cbl\base.

The following is the PTCCBLAE.CBL copybook.

```
*01  CBLAE.

NOCLN      02  CBLAE-PRCSNAME          PIC X(12)   VALUE SPACE.

NOCLN      02  CBLAE-COMMIT-FLAG       PIC X(1)    VALUE SPACE.

               88 AE-COMMITS-SUCCESS               VALUE 'B'.

               88 AE-COMMITS-ALL                   VALUE 'C'.

           02  CBLAE-PARMS.

               03  CBLAE-PARM-CNT      PIC 9(4)                COMP.

               03  CBLAE-PARM-ENT      OCCURS 500 TIMES.

                   05 CBLAE-STATEREC   PIC X(15).

                   05 CBLAE-FIELDNM    PIC X(18).

                   05 CBLAE-DATA-PTR               POINTER.

                   05 CBLAE-LENGTH     PIC 9999                COMP.
```

```
                    05 CBLAE-SCALE      PIC 99                  COMP.
NOCLN               05 CBLAE-TYPE       PIC X.
                        88  CBLAE-TYPE-CHAR         VALUE 'C'.
                        88  CBLAE-TYPE-SMALLINT     VALUE 'S'.
                        88  CBLAE-TYPE-INT          VALUE 'I'.
                        88  CBLAE-TYPE-DEC          VALUE 'P'.
                        88  CBLAE-TYPE-DATE         VALUE 'D'.
                        88  CBLAE-TYPE-TIME         VALUE 'T'.
                        88  CBLAE-TYPE-TIMEONLY     VALUE 'V'.
                        88  CBLAE-TYPE-NUMERIC      VALUE 'S' 'I' 'P'.
```

Data Transfer Process Between COBOL Programs and Application Engine Programs

To interface between COBOL programs and Application Engine programs, the process uses a file to pass parameters from COBOL to the Application Engine program. This file is owned by the process and has the *prm* extension. The location of the file is determined by the following:

- If an application server root directory is defined, the file resides in the output directory of that particular process instance.
- If the output directory on the application server is not defined, the file resides in the default output directory of the Process Scheduler domain..
- If neither one of the above is defined, the file is written to the default temp directory.

Assigning Copybook Values

To assign values to the calling COBOL program's copybook to be passed as parameters into the state records of the called Application Engine program:

- Identify the fields in your COBOL program that contain the values you want to pass to the Application Engine program.
- Load the PTCCBLAE.CBL copybook with the state record name, field name, field length (this should be the size of the field not the size of the contents), the scale (decimal places if any), and set the field type.
- Call the PTPSETAD program to set the pointer in PTCCBLAE.CBL to the host programs variable.
- Set the variable AE-COMMIT-FLAG to either AE-COMMITS-ALL or AE-COMMITS-SUCCESS.

AE-COMMITS-ALL means that the Application Engine program commits as specified in the program. AE-COMMITS-SUCCESS means that the Application Engine program ignores all commits and performs one commit at the end of successful execution.

Example of Loading Values from PTPTSTAE.CBL Sample Program

Make sure the calling COBOL program has successfully connected to the database before calling the PTPCBLAE copybook, and ensure that the calling program is not running through a RemoteCall function.

The following code shows an example of how to load values from the copybook:

```
MOVE 0 TO CBLAE-PARM-CNT OF CBLAE


            ADD  1 TO CBLAE-PARM-CNT OF CBLAE

            MOVE 'QE_CBLAETST_AET' TO CBLAE-STATEREC

                   OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

            MOVE 'DESCR' TO CBLAE-FIELDNM

                   OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

            MOVE 30 TO CBLAE-LENGTH

                   OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

            MOVE 0 TO CBLAE-SCALE

                   OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

            SET CBLAE-TYPE-CHAR OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

                   TO TRUE

            CALL 'PTPSETAD' USING  CBLAE-DATA-PTR

                   OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

                   W-DESCR OF W-WORK


            ADD  1 TO CBLAE-PARM-CNT OF CBLAE

            MOVE 'QE_CBLAETST_AET' TO CBLAE-STATEREC

                   OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

            MOVE 'QE_AE_INT_7' TO CBLAE-FIELDNM
```

```
                OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

           MOVE 2 TO CBLAE-LENGTH

                OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

           MOVE 0 TO CBLAE-SCALE

                OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

           SET CBLAE-TYPE-SMALLINT

   OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

                TO TRUE

           CALL 'PTPSETAD' USING  CBLAE-DATA-PTR

                OF CBLAE (CBLAE-PARM-CNT OF CBLAE)

                W-SMINT OF W-WORK


*

       DA000-CALL-AE SECTION.

       DA000.

*

    MOVE 'QE_AETESTPRG' TO CBLAE-PRCSNAME OF CBLAE

           SET AE-COMMITS-ALL TO TRUE


       CALL 'PTPCBLAE' USING SQLRT CBLAE.

       CALL-AE-EXIT.

       EXIT.
```

Sample of the Communication Area of PTPBLAE.CBL

If the called Application Engine program updated the state records or fields that were passed by PTPCBLAE, they fields or records are stored in the calling program's local variables as identified by PTPSETAD.

```
* PTCCBLAE - Communication area for PTPCBLAE                    *
```

```
*01  CBLAE.
NOCLN   02   CBLAE-PRCSNAME   PIC X(12)   VALUE SPACE.
*      Name of AE program to be called.
NOCLN   02   CBLAE-COMMIT-FLAG   PIC X(1)   VALUE SPACE.
*      Flag to determine which of the following commits to make.
           88 AE-COMMITS-SUCCESS         VALUE 'B'.
*      No in-process commit; if successful, then commit occurs.
           88 AE-COMMITS-ALL         VALUE 'C'.
*      Commits occur when defined in the AE program.
         02  CBLAE-PARMS.
           03  CBLAE-PARM-CNT   PIC 9(4)COMP.
*      Counter of the number of state records passed.
           03  CBLAE-PARM-ENT   OCCURS 500 TIMES.
*      Maximum value of state record entries.
             05 CBLAE-STATEREC   PIC X(15).
*      State record name.
             05 CBLAE-FIELDNM   PIC X(18).
*      Field name.
             05 CBLAE-DATA-PTR   POINTER.
*      Pointer to your own working storage area.
             05 CBLAE-LENGTH   PIC 9999   COMP.
*      Field length of defined state record.
             05 CBLAE-SCALE   PIC 99   COMP.
*      Number of decimal places.

NOCLN        05 CBLAE-TYPE      PIC X.
```

```
*      Field data type.

                88  CBLAE-TYPE-CHAR         VALUE 'C'.

                88  CBLAE-TYPE-SMALLINT     VALUE 'S'.

                88  CBLAE-TYPE-INT          VALUE 'I'.

                88  CBLAE-TYPE-DEC          VALUE 'P'.

                88  CBLAE-TYPE-DATE         VALUE 'D'.

                88  CBLAE-TYPE-TIME         VALUE 'T'.

                88  CBLAE-TYPE-TIMEONLY     VALUE 'V'.

                88  CBLAE-TYPE-NUMERIC      VALUE 'S' 'I' 'P'.
```

Handling COBOL Errors

If your COBOL program needs error handling, try the following process:

1. Add a field (return code) to your state record.
2. Initialize the field to a negative value.
3. Pass the value into the Application Engine program.
4. At the successful completion of the Application Engine program, change the field value to a positive value.
5. Check for that value in your COBOL program.

CHAPTER 8

Tracing Application Engine Programs

This chapter provides overviews of tracing Application Engine programs and trace results and discusses how to:

- Enable Application Engine tracing.
- Locate trace files.

Understanding Tracing Application Engine Programs

You can set the following traces to monitor the performance of Application Engine programs:

- Application Engine step trace.
- Application Engine SQL trace.
- Application Engine statement timings trace.
- Database optimizer trace.

Note. The general PeopleTools SQL and PeopleCode traces also apply to Application Engine programs.

Understanding Trace Results

This section discusses:

- Trace file sections.
- Step trace.
- SQL trace.
- Statement timings trace.
- Database optimizer trace.

Trace File Sections

At the top of each trace, useful information helps you to identify the PeopleTools version, the database name, and the database platform type.

SQL Counts and Timings Section

The first section of a trace file is the SQL section. It records the performance of application-specific SQL. The trace values appear within a series of columns and sections. The following table describes each column within the first section of the trace file.

Column	Description
SQL Statement	Application Engine SQL actions and stored SQL objects always have a statement ID. The SQL Statement column shows the statement ID, so that you can attribute trace values to individual SQL statements. In the case of SQLExec SQL, a portion of the SQL statement appears in the first column, to help you identify it. For SQL objects, use the TraceName property in the Create SQL so that you can uniquely identify it in the traces.
Compile Column	This column shows how many times the system compiled a SQL statement and how long it took. The term *compiled* refers to the SQL statement being sent to the database to be parsed and optimized, and it also includes the time required for the first resolution of any PeopleSoft meta-SQL.
Execute Column	This column shows how many times the system executed the SQL statement and the time consumed doing so. The term *executed* refers to the system sending the compiled SQL to the database server to be run against the database.
Fetch Column	This column applies to Select statements. It shows how many rows your program fetched from the database and how much time this consumed. The system must first execute a Select statement against the database to find the relevant rows and generate an active set. After the set exists, the program must still fetch the rows. Some database APIs have buffered fetches, which means that the fetch may include more than one row. This makes subsequent fetches free until the buffer becomes empty.
Total Column	This column shows the sum of the compile, execute, and fetch times of the SQL statement. Some database APIs may defer a compile to the execute phase, or defer an execute to the first fetch operation.
PeopleCode SQL	This subsection is for SQL executed from PeopleCode actions. Compile counts and times for such SQL is included in execute count and times, because you do not explicitly control the ReUse feature. To determine whether ReUse is ocurring, you must do a program run after enabling the generic PeopleTools trace for SQL statements, API calls, and so on. As a starting point, use a trace value of 31.
Application Engine SQL	This subsection reveals the time attributed to PeopleSoft Application Engine overhead that is not directly related to the SQL within your program. For example, the values in this section represent the SQL generated for checkpoints, commits, and so on. If there are Commit statements without checkpoints, it indicates that restart has been disabled, or a restartable program has called a nonrestartable program. If the time consumed performing a checkpoint or committing seems more than expected, you should try to reduce it if possible by setting the commit frequency of the steps containing Do loops.
AE Program: *program_name*	This subsection shows SQL actions for a particular program. The action properties that affect performance are flagged. For example, BulkInsert. ReUse is not flagged because it is self-evident when the Execute count is higher than the compile count.

Note. When you run a SQL trace at the PeopleSoft Application Engine level and the PeopleTools level simultaneously, you may see misleading results. Extra overhead gets added to the overall SQL timings by the PeopleTools trace. Tracing SQL at the PeopleSoft Application Engine level (-TRACE) adds to the non-SQL times because PeopleTools writes the trace data after timing the SQL.

PeopleCode Actions Section

The second section of the trace file, or PeopleCode section, records the performance associated with all the PeopleCode actions in your program. The following table describes each column in this section:

Column	Description
PeopleCode	This column contains the names of the PeopleCode actions in your program.
Call	This column shows how many times each PeopleCode action gets called during the program run.
Non-SQL	This column shows the time spent by your PeopleCode actions that does not involve any SQL.
SQL	This column shows the time spent by your PeopleCode actions executing SQL. The SQL time total should be similar to that of the PeopleCode SQL subsection in the first section of the trace file.
Total	The total indicates the cumulative amount of time spent in the action.

Note. The system rounds to the first decimal place (tenths), but only after calculating the sum of each action time.

PeopleCode Built-ins and Methods Section

The third section of the trace file contains either a list or summary of PeopleCode built-ins and methods used. To get a list of built-ins and methods, you must enable the PeopleCode detail timings in addition to the statement timings trace.

If a method or built-in function takes a large amount of time, you may want to consider alternatives. For example, if array processing dominates your runtime, consider inserting the data into temporary tables and performing the processing on tables in the database.

Summary Data

The fourth section of the trace file contains summary data. The values in this section reveal an overview of the program run without drilling down too far into details.

The following table describes the values that appear in this section:

Column	Description
Total run time	This value presents the overall amount of time a program required to complete from start to finish.
Time in application SQL	This value represents the time that your program spent executing SQL. The value includes SQL executed by both PeopleCode and SQL actions.

Column	Description
Percent time in application SQL	This value represents the percentage of time spent executing SQL compared to the entire program run.
Time in PeopleCode	This value represents the time that your program spent executing PeopleCode. Time in PeopleCode excludes SQL executed from within PeopleCode.
Percent time in PeopleCode	This value represents the percentage of time spent executing PeopleCode compared to the entire program run.
Total time in Cache	This value represents the amount of time your program spent retrieving objects from the cache or refreshing the cache. Time in cache includes all memory cache access, file cache access, and SQL executed to load managed objects, such as Application Engine program components, metadata, and so on. Time varies according to where PeopleSoft Application Engine finds an object. For instance, retrieving an object that the system cached during a previous run is faster than retrieving it from the database.
Number of calls to Cache	This value represents the actual number of calls your program made to the cache. The number of calls to the cache remains constant for the same Application Engine program processing the same data.

Environment Information Section

The fifth section of the trace file contains environment information specific to PeopleSoft Application Engine. If programs appear to be performing poorly, check the trace value that you have set.

Each trace produces an unavoidable degree of overhead. As such, the more traces you have enabled, the more likely you are to see degraded performance.Run only the traces you need. This section of the trace file shows you information about the following:

- SQL trace
- PeopleCode trace
- Application Engine trace
- Application Engine DbFlags (%UpdateStats)

Step Trace

The step trace reports each step name that your program executes and in what order. Associated with each step is a timestamp, the Do action level, and the action type.

The trace shows the steps that execute within a called section by indented formatting. For example, a step that executes within a called section is preceded by two dots (..), while other steps are preceded by only one dot.

SQL Trace

The SQL trace report shows formatted SQL processes, including commits, rollbacks, and restarts. You can also view the buffers associated with each SQL statement. Use the SQL trace to spot errors in your SQL and to view your commit strategy.

Statement Timings Trace

The Application Engine statement timing trace report is similar to a COBOL timings trace, in which you monitor the execution of COBOL programs for performance evaluations. This trace enables you to gather performance information to determine program bottlenecks. Once bottlenecks have been identified,you might be able to modify your program to run more efficiently, or you may want to change the database schema and configuration to optimize the execution of your program.

The statement timings trace is invaluable for tuning an Application Engine program. It may also be useful as a default trace level for all production runs, to provide a metric for long-term performance trends.

By examining all of the figures in this trace, you can identify areas of your program that are not running as efficiently as possible. For instance, if compile counts are high, you can reduce the numbers by using the PeopleSoft Application Engine reuse feature. If inserts appear to be running slow and you have many of them, you can increase the performance by using the PeopleSoft Application Engine bulk insert feature. Each value in the trace, including cumulative totals, appears in a form rounded to the nearest tenth of a second, but totals are calculated using nonrounded timings.

You can write this trace to a file, or you can write the results to tables. Either way, timings trace overhead is minimal. Internal testing reveals that the Application Engine trace has an overhead between 2 percent and 5 percent of total runtime.

By storing timings information in a table, you can store historical data in the database, which enables you to produce reports that aid in trend analysis, allow ad hoc SQL queries for longest running statements, and so on. With the timings data stored in the database, you can manipulate and customize reports to show only the metrics in which you are most interested.

You can use third-party tools to query and present the data in such ways as detailed graphical representations of your program's performance. You can also implement alarms if the performance of a program reaches a specified maximum value in a particular area such as SQL compile time.

Note. PeopleSoft Application Engine does not write the timings trace to a table for programs invoked by the CallAppEngine PeopleCode function. To write to a table, a process instance is required, and programs invoked by CallAppEngine are not assigned a process instance.

The statement timings trace populates the following tables.

- PS_BAT_TIMINGS_LOG (Parent)

  This table stores general information for a program run.

- PS_BAT_TIMINGS_DTL (Child)

  This table stores details associated with a program run, such as the execute count, fetch time, and so on.

- PS_BAT_TIMINGS_FN

  This table stores PeopleCode detail timings information.

PeopleSoft provides BATTIMES.SQR as an example of the type of reports you can generate to reflect the information stored in the BAT_TIMINGS tables. You can produce a summary report for all the programs for a specific run control ID, or you can get detail data for a specific process instance.

To invoke the BATTIMES.SQR report through PeopleSoft Process Scheduler:

1. Select PeopleTools, Process Scheduler, Batch Timings.

   The Batch Timings page appears.

2. From the Report Type drop-down list box, select *Detail* or *Summary.*

3. In the Batch Timings For group box, enter the run control ID for summary reports, and enter the process instance for detail reports.
4. When you have made the appropriate selections, click the Run button.

To view batch timings using Process Monitor:

1. Select PeopleTools, Process Scheduler, Process Monitor.
2. Locate the program run associated with the current trace.
3. Click the Job Details button.
4. On the Process Detail dialog box, click the Batch Timings link.

   PeopleCode detail timings do not appear. They appear only in the file format.

Database Optimizer Trace

The database optimizer trace reveals the execution or query plan for the SQL that your Application Engine program generates. Each SQL statement gets traced only once. You can write the trace to a file or a table.

How you view the results of this trace depends on the relational database management system (RDBMS) that you're currently using. For instance, on some platforms, only the trace-to-file option is available, whereas on others only the trace-to-table option is available. The following table shows the options available for each of the platforms PeopleSoft supports:

RDBMS	Output
Microsoft SQL Server	File and table
DB2 for OS/390	Table
DB2 for UDB (AIX, Sun Solaris, Microsoft Windows NT)	Table
Oracle	File and table
Informix	File
Sybase	N/A

Note. PeopleTools does not collect optimizer data for SQL originating from PeopleCode actions, except if you run Oracle and Informix and use file output. In this case, the system traces all SQL that executes after the first SQL action executes.

Microsoft SQL Server

When you output the trace to a file, PeopleSoft Application Engine writes the optimizer trace to the following location: %TEMP%\psms<queueid><spid>.trc. To read the trace, you use the SQL Server Profiler utility.

Note. The trace file is written to the server directory when you've specified the trace on the client. If the client has %Temp% set to a drive or directory that does not exist on the server, PeopleSoft Application Engine does not generate a trace file.

When you output to a table, PeopleSoft Application Engine writes the trace data to the dbo.PS_OPTIMIZER_TRC table. PeopleTools creates the table automatically when you run the trace for the first time. The trace data written to the table is identical to the data appearing in the optimizer trace file.

You also need to use the SQL Server Profiler utility to view the optimizer results. To view the populated trace table, specify the current server and database on the Source Table dialog box. The Owner value must be *dbo,* and the Table value *PS_OPTIMIZER_TRC*.

In the trace, you find information regarding text, duration, andstart time for the following:

- Execution plans.
- Remote procedure calls.
- Insert statements (Update, Delete, and Select statements).
- PeopleSoft-generated user events associating trace data with a PeopleSoft SQL identifier.

If the Application Engine program terminates while you're using this trace option, check that PeopleSoft Application Engine was not tracing a SQL statement at the moment the program terminated. If it was tracing a SQL statement at that time, you must manually kill the trace. Otherwise the trace thread on the server continues to run and lock the trace file, and each time that server process ID (SPID) gets reused by the server, new information is appended to the locked trace file.

To stop the trace manually, submit the following command from Query Analyzer:

```
xp_trace_destroyqueue queueid
```

The *queueid* in the file name %TEMP%\psms_*queueid_spid*.trc is the ID corresponding to the queue used for the first SQL statement that the system profiled. Because this trace is only designed to trace Application Engine SQL (not PeopleTools SQL), the queue is closed after every statement profiled. Therefore the queue that must be destroyed may not be the queue ID used in the trace file.

Note. If the %TEMP% variable is set to a location that does not exist, PeopleSoft Application Engine does not generate a trace file.

Oracle

When outputting the trace to a file, PeopleSoft Application Engine writes the trace file to the default Oracle trace directory specified on the database server. To read the trace file, use the TKPROF utility.

To output the trace to a table on Oracle, a PLAN_TABLE table must exist, and the statement_id must be of type VarChar2(254), instead of VarChar2(30).

When outputting to a table, PeopleSoft updates the trace rows as follows:

- EXPLAIN PLAN SET STATEMENT_ID: PeopleSoft updates the STATEMENT ID column:

```
EXPLAIN PLAN SET STATEMENT_ID = ApplId.Section.Step.Type FOR sqlstmt
```

- PLAN_TABLE's REMARKS column: PeopleSoft updates the REMARKS column:

```
PLAN_TABLE's REMARKS column = 'ProcessInstance-RunControlId(QueryNo)'
```

queryno is a count of how many SQL statements have been traced up to a particular point.

Note. When tracing to a table with Oracle, PeopleSoft does not perform optimizer traces on %UpdateStats and %TruncateTable unless the latter resolves into a Delete statement. Alternatively, outputting Oracle's TKPROF utility to file handles both the Analyze and Truncate commands.

Informix

For Informix, you can only output the trace to a file. The trace file location depends on the operating system on which your database server runs.

- UNIX.

  For UNIX, PeopleSoft Application Engine writes the plan to the sqexplain.out file. If the client program runs on the same machine as the database server, the sqexplain.out file appears in the current directory. When the current database is on another computer, the sqexplain.out file gets written to the PeopleSoft owner's directory on the remote host.

- Microsoft Windows NT.

  For Microsoft Windows NT, PeopleSoft Application Engine writes the plan to the following file: INFORMIXDIR%\sqexpln\username.out.

DB2 for OS/390

For DB2 for OS/390, you can only output the optimizer trace to a table. PeopleSoft has implemented the following to facilitate this trace:

- PeopleSoft selects the maximum query number from the PLAN_TABLE table, increments it by 1000 to avoid clashing with other processes, and then increments it by 1 for every SQL statement traced.
- PeopleSoft sets the SET REMARKS parameter to the following value: ApplId.Section.Step.Type-RunControlId(ProcessInstance)

Note. Before using the Database Optimzer Trace, you must first create a DB2 PLAN_TABLE. Refer to your DB2 UDB for OS/390 and z/OS Administration Guide for the proper format and instructions on creating the PLAN_TABLE.

DB2 for UNIX

For DB2 for UNIX, you can only output the optimizer trace to a table. To facilitate this trace for DB2/UNIX, PeopleSoft has implemented:

EXPLAIN ALL SET QUERYNO =ProcessInstance SET QUERYTAG = 'Section.Step' FOR *sql stmt*

Note. Before using the Database Optimizer Trace, you must first create the DB2 explain tables.

Database Optimizer Trace and Performance

While the database optimizer trace is enabled, performance may be affected. Typically, you turn on this trace only when you are collecting detailed performance metrics. When you are not tuning your performance, turn off the optimizer trace.

To prevent an administrator, or perhaps a user, from unwittingly turning the optimizer trace on or leaving it on after doing performance tuning, you can disable the database optimizer trace for an entire database.

For example, suppose you have a production and a development database, you might want to enable the optimizer trace for the development database while disabling the optimizer trace for the production database.

On the PeopleTools Options page, clear the Allow DB Optimizer Trace option to disable the optimizer trace for the database.

Enabling Application Engine Tracing

By default, all Application Engine traces are turned off. To see a trace or a combination of traces, set trace options prior to executing the program.

This section discusses how to:

- Set command line options.
- Set parameters in server configuration files.
- Set options in Configuration Manager.

Setting Command Line Options

The command line option is available for Microsoft Windows NT and UNIX, but it is not available when calling Application Engine programs from PeopleCode.

To enable tracing from the command line, specify the −TRACE option within the command line that you submit to PSAE.EXE. For example,

```
n:\pt840\bin\client\winx86\psae.exe -CT MICROSFT -CD PT800GES -CO PTDMO?
 -CP PTDMO -R PT8GES -AI AETESTPROG −I 45 -TRACE 2
```

The following table describes the available TRACE option parameter values:

Value	Description
0	Disables tracing.
1	Initiates the Application Engine step trace.
2	Initiates the Application Engine SQL trace.
4	Initiates the trace for dedicated temporary table allocation to an Application Engine trace (AET) file. You can trace how the system allocates, locks, and releases temporary tables during program runs.
128	Initiates the statement timings trace to a file, which is similar to the COBOL timings trace to a file.
256	Initiates the PeopleCode detail to the file for the timings trace.
1024	Initiates the statement timings trace, but stores the results in the following tables: PS_BAT_TIMINGS_LOG and PS_BAT_TIMINGS_DTL.
2048	Requests a database optimizer trace file.

Value	Description
4096	Requests a database optimizer to be inserted in the Explain Plan table of the current database.
8192	Sets a trace for PeopleSoft Integration Broker transform programs.

To specify traces on the command line, you enter the sum of the desired trace options. This is similar to adding the trace values using PSADMIN, such as the COBOL statement timings or the SQL statement trace value. To specify a combination of traces, enter the sum of the corresponding trace values. For example, to enable the step (1), the SQL (2), and the statement timings (128) traces, you would enter 131—the sum of 1, 2, and 128.

To disable tracing, explicitly specify −TRACE 0. If you don't include the −TRACE flag in the command line, PeopleSoft Application Engine uses the value specified in the Process Scheduler configuration file or in Configuration Manager. Otherwise, the command-line parameters override any trace settings that may be set in Configuration Manager.

Setting Parameters in Server Configuration Files

You can also enable traces in the configuration files for both the application server and the PeopleSoft Process Scheduler server.

For programs invoked by PeopleCode and run on the application server, set the TraceAE parameter in the Trace section of the Application Server configuration file (PSAPPSRV.CFG). You can use PSADMIN to set this parameter.

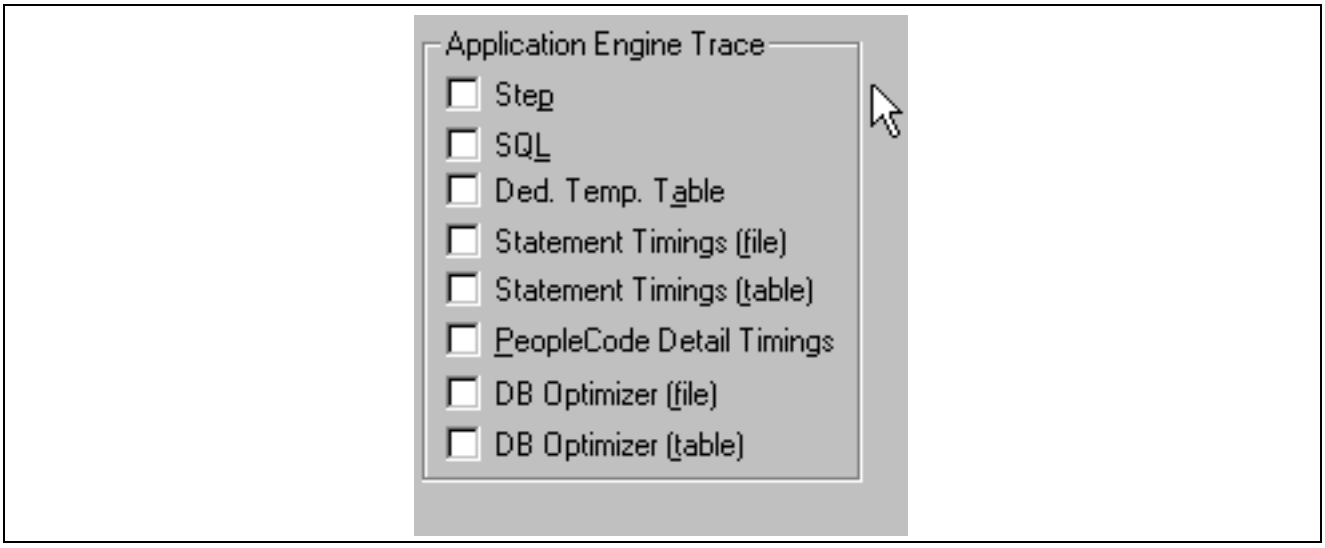
In the PeopleSoft Process Scheduler configuration file, set the TraceAE parameter in the Trace section to indicate a level of tracing. You can use PSADMIN to set this parameter.

This option is available on Microsoft Windows NT and UNIX, and applies only to Application Engine programs invoked in batch mode.

Note. The TraceFile parameter does not specify the location of the Application Engine trace file; it applies only to the generic PeopleTools SQL and PeopleCode traces.

Setting Options in PeopleSoft Configuration Manager

For processes running on a Microsoft Windows workstation, you can set trace options using PeopleSoft Configuration Manager. This procedure is valid only if you are running Application Engine programs on a Microsoft Windows workstation—the development environment.



Application Engine Trace check boxes

To set Application Engine traces:

1. Start Configuration Manager, and select the Trace tab.
2. Select appropriate trace options.

   You can select any combination of the options.
3. Click either the Apply or OK buttons to set trace options.

Locating Trace Files

Where you look for the generated trace file depends on how you invoked the program and the operating system on which the program runs, as shown in the following table:

Location Where the Program Was Initiated	Trace File Location
Microsoft Windows workstation	Look for the trace file in %TEMP%\PS\<db name>.
PeopleCode	Look for the trace file in %TEMP%\PS*db_name* on Microsoft Windows NT and in *PS_HOME*/log/\<db name> on UNIX and Linux systems.
Command line	Look for the trace file in the directory specified in the Log/Output field in the PS_SERVER_CFG file.
PeopleSoft Process Scheduler	Look for the trace file in a subdirectory of the directory specified in the Log/Output field in the PS_SERVER_CFG file.

When a program includes a process instance, PeopleSoft Application Engine names the trace file according to the following convention: AE_*Program_name_Process_Instance*.AET. When the program does not include a process instance, PeopleSoft Application Engine names the file according to this convention: AE_*Date/Time_Stamp_OS*_PID.AET. The date and time stamp is in the format month, day, hour, minute, second, with two values for each date element and no punctuation between elements. For example, August 12 at 5:09 p.m. and 30 seconds would be 0812170930.

Note. For an Application Engine program running on a server, PeopleTools writes the generic PeopleTools trace for SQL and PeopleCode trace files to the same directories as the AET traces. The prefix of the trace file name is also the same, and the suffix is trc. On the Windows workstation, the trace is written to the "People Tools Trace File" specified in the Trace folder of PeopleSoft Configuration Manager.

CHAPTER 9

Using Temporary Tables

This chapter provides an overview of temporary tables and discusses how to:

- Create temporary table instances.
- Manage temporary table instances.
- Make external calls.
- View temporary table usage.

Understanding Temporary Tables

Because Application Engine programs run in batch mode, multiple instances of the same program often execute in parallel. When this happens, there is a significant risk of data contention and deadlocks on tables. To avoid this, you can dedicate specific instances of temporary tables for each program run.

You can also use temporary tables to improve performance. For example, if you find that, multiple times during a run, the program accesses a small subset of rows from a much larger table, you can insert the necessary rows into a temporary table as an initialization task. Then the program accesses the data residing in the smaller temporary table rather than the large application table. This technique is similar to reading the data into an array in memory, except that the data never leaves the database, which is an important consideration when the program employs a set-based processing algorithm.

Any number of programs, not just Application Engine programs, can use the temporary table definitions. When you specify a temporary table on the Temp Tables tab in the Application Engine program properties, PeopleSoft Application Engine automatically manages the assignment of temporary table instances. When Application Engine manages a dedicated temporary table instance, it controls the locking of the table before use and the unlocking of the table after use.

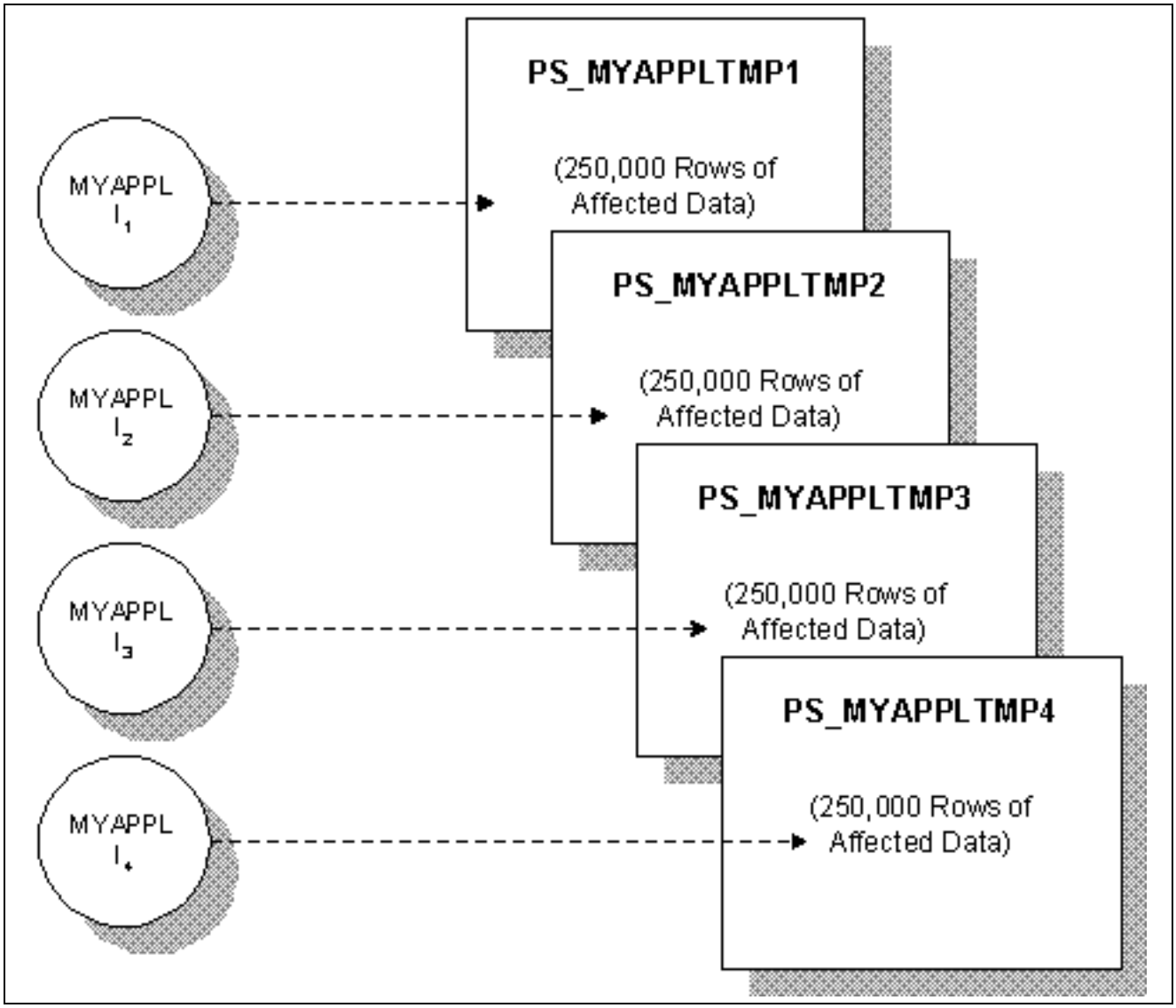
Parallel Processing

Parallel processing is used when considerable amounts of data must be updated or processed within a limited amount of time, or batch window. In most cases, parallel processing is more efficient in environments containing multiple CPUs and partitioned data.

To use parallel processing, partition the data between multiple concurrent runs of a program, each with its own dedicated version of a temporary table (for example, PS_MYAPPLTMP). If you have a payroll batch process, you could divide the employee data by last name. For example, employees with last names beginning with A through M get inserted into PS_MYAPPLTMP1; employees with last names beginning with N-Z get inserted into PS_MYAPPLTMP2.

To use two instances of the temporary table, you would define your program (say, MYAPPL) to access to one of two dedicated temporary tables. One execution would use A-M and the other N-Z.

The Application Engine program invokes logic to pick one of the available instances. After each program instance gets matched with an available temporary table instance, the %Table meta-SQL construct uses the corresponding temporary table instance. Run control parameters passed to each instance of the MYAPPL program enable it to identify which input rows belong to it, and each program instance inserts the rows from the source table into its assigned temporary table instance using %Table. The following diagram illustrates this process:



Multiple program instances running against multiple temporary table instances

There is no simple switch or check box that enables you to turn parallel processing on and off. To implement parallel processing, you must complete the following set of tasks. With each task, you must consider details regarding your specific implementation.

1. Define and save temporary table records in PeopleSoft Application Designer.

   You don't need to run the SQL Build process at this point.

2. In PeopleSoft Application Engine, assign temporary tables to Application Engine programs, and set the instance counts dedicated for each program.

Employ the %Table meta-SQL construct so that PeopleSoft Application Engine can resolve table references to the assigned temporary table instance dynamically at runtime.

3. Set the number of total and online temporary table instances on the PeopleTools Options page.
4. Build temporary table records in PeopleSoft Application Designer by running the SQL Build process.

Creating Temporary Table Instances

This section provides an overview of temporary table instances and discusses how to:

- Define temporary tables.
- Set the number of temporary table instances.
- Build table instances.

Understanding Temporary Table Instances

To run processes in parallel, you need to enable multiple instances of the same temporary table. You use the PeopleTools Options page to set the number of temporary table instances for Application Engine processes started online from the PeopleCode CallAppEngine function.

This global setting is separate from the instance count setting for a particular program. To use a temporary table with a specific program, you assign the table to the program and set the number of instances created when a particular program is run.

Key Fields for Temporary Tables

To take advantage of multiple instances of a temporary table, use the Temporary Table record type.

Insert the PROCESS_INSTANCE field as a key on any temporary tables that you intend to use with PeopleSoft Application Engine. PeopleSoft Application Engine expects Temporary Table records to contain the PROCESS_INSTANCE field.

Note. When all instances of a temporary table are in use and the Continue runtime option on the Program Properties dialog box Temp Table tab is selected, PeopleTools inserts rows into the base table using PROCESS_INSTANCE as a key. If you do not include PROCESS_INSTANCE as a key field in a temporary table, select the Abort Temp Table tab runtime option.

Temporary Table Performance Considerations

When you run batch processes in parallel, there is a risk of data contention and deadlocks on temporary tables. To avoid this, PeopleSoft Application Engine has a feature that enables you to dedicate specific instances of temporary tables for each process. When PeopleSoft Application Engine manages a dedicated temporary table instance, it controls the locking of the table before use and the unlocking of the table after use.

When you decide on the number of instances for temporary tables for a process, you must take into consideration the number of temporary tables that the process uses. The more instances you have the more copies of the temporary tables you will have on your system. For example, if a process uses 25 temporary tables and you have 10 instances for a process, you will have 250 temporary tables on your system.

On the other hand, if you are running a process in parallel and all of the dedicated temporay table instances are in use, this slows down the performance of the process. So, you will need to find a balance that works for your organization.

If you need more temporary table instances after you've entered production, you must rebuild all of your temporary tables so that the database reflects the proper inventory of instances. While the build process runs, users can't access the database. Because of this, spend time deriving adequate estimates as to the number of temporary tables required.

A physical table within the database, named PS_AEONLINEINST, stores online temporary table instance usage. If you notice performance issues related to online Application Engine program runs, enable the Application Engine SQL and Timings trace.

If the following SQL command requires more time than normal to complete, this is a good indication that there aren't enough online temporary instances defined on the PeopleTools Options page.

```
UPDATE PS_AEONLINEINST ...
```

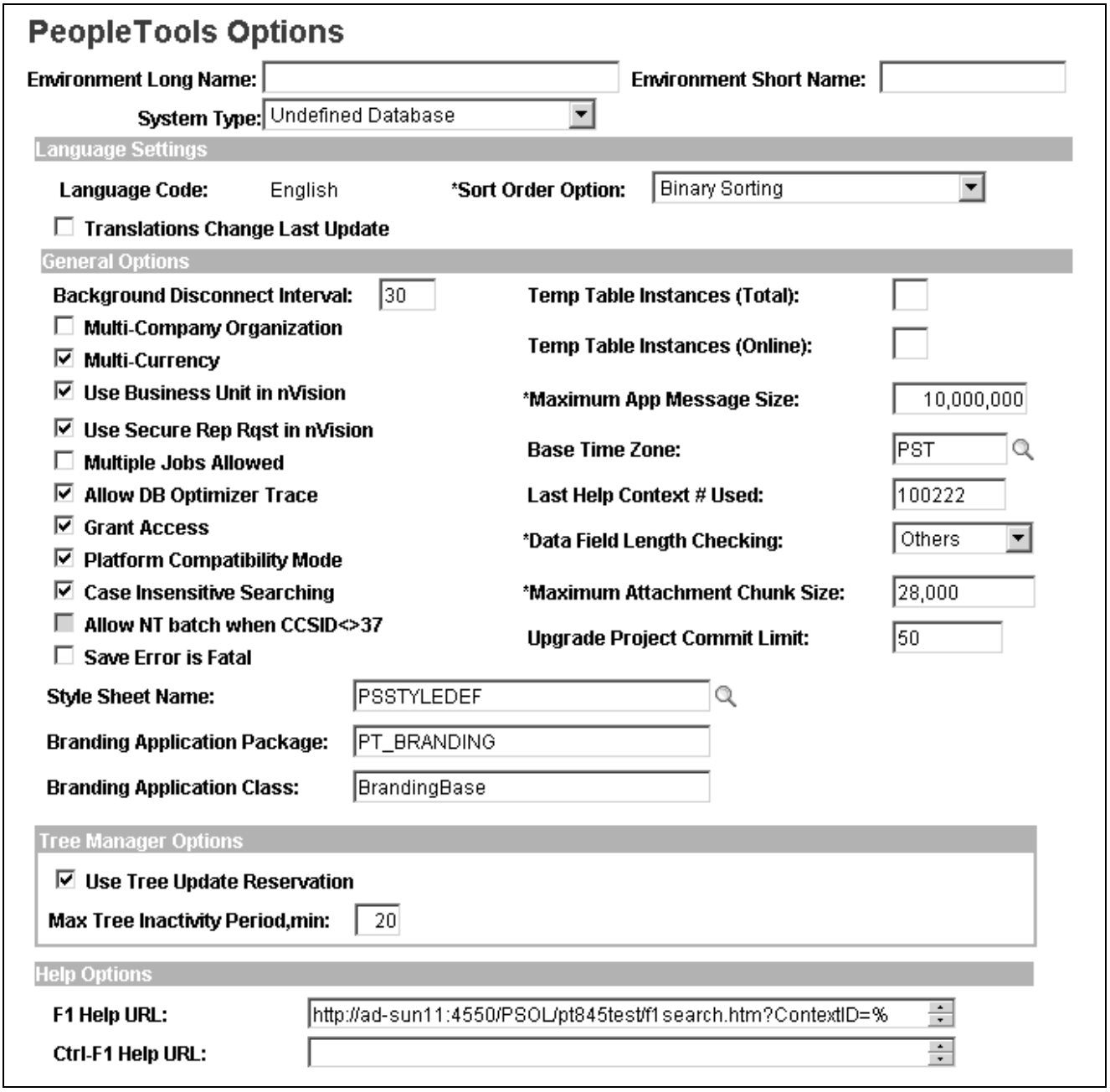
Defining Temporary Tables

To define a temporary table:

1. In PeopleSoft Application Designer, select File, New.
2. Select Record from the New Definition dialog box.
3. Select Insert, Field, and insert the PROCESS_INSTANCE field.
4. Select the Record Type tab and select the Temporary Table option.

Setting the Number of Temporary Table Instances

Select PeopleTools, Utilities, Administration, PeopleTools Options to access the PeopleTools Options page.



PeopleTools Options page

The system determines the total available number of temporary table instances for a base table according to the settings for total and online instances that you make on this page.

Temp Table Instances (Total)(temporary table instances [total]) The difference between the total and online numbers is your EPM-managed tables. If you are not using PeopleSoft EPM, the total and online numbers should be the same.

Temp Table Instances (Online)(temporary table instances [online]) Enter the number of temporary table instances for Application Engine processes started online from the PeopleCode CallAppEngine function. In general, the number you enter should be relatively small (less than 10), so that extra instances do not affect performance.

PeopleSoft Application Engine uses this value to identify a range of temporary tables devoted to programs called by the CallAppEngine function. A randomizing algorithm balances the load for the online process that gets assigned to a temporary table devoted to online program execution.

Building Table Instances

The system builds temporary table instances at the same time it builds the base table for the record definition. When the system builds a table (as in, Build, Current Object) and the record type is Temporary Table, it determines the total number of instances of the temporary table based on the settings that you made on the PeopleTools Options page.

The system creates a maximum of 99 temporary table instances, even if the sum exceeds 99 for a particular temporary table.

The naming convention for the temporary table instances is as follows: BaseTableName_Number, where *Number* is a number between 1 and 99, as in PS_TEST_TMP23.

Note. You can take advantage of database-specific features such as table spaces and segmentation. For instance, you may want to use the Build process to generate a data definition language (DDL) script, then fine-tune the script prior to execution, or you could place different sets of temporary tables on different table spaces according to instance number.

Managing Temporary Table Instances

This section provides an overview of temporary table instance numbers and discusses how to:

- Assign temporary tables to programs.
- Adjust meta-SQL.

Understanding Temporary Table Instance Numbers

You use the Temp Tables tab in the Program Properties dialog box to manage the number of different batch or dedicated temporary tables required for each program definition and the number of instances of each. You select all the necessary temporary table records to meet the needs of your program‘s logic.

Note. You must set the instance count on the Temp Tables tab prior to building the tables in PeopleSoft Application Designer.

Regardless of the instance counts value in the Application Engine program properties or on the PeopleTools Options page, make sure that you have the appropriate records assigned to the appropriate programs. You also need to ensure that the SQL inside your Application Engine program contains the correct usage of the %Table construct.

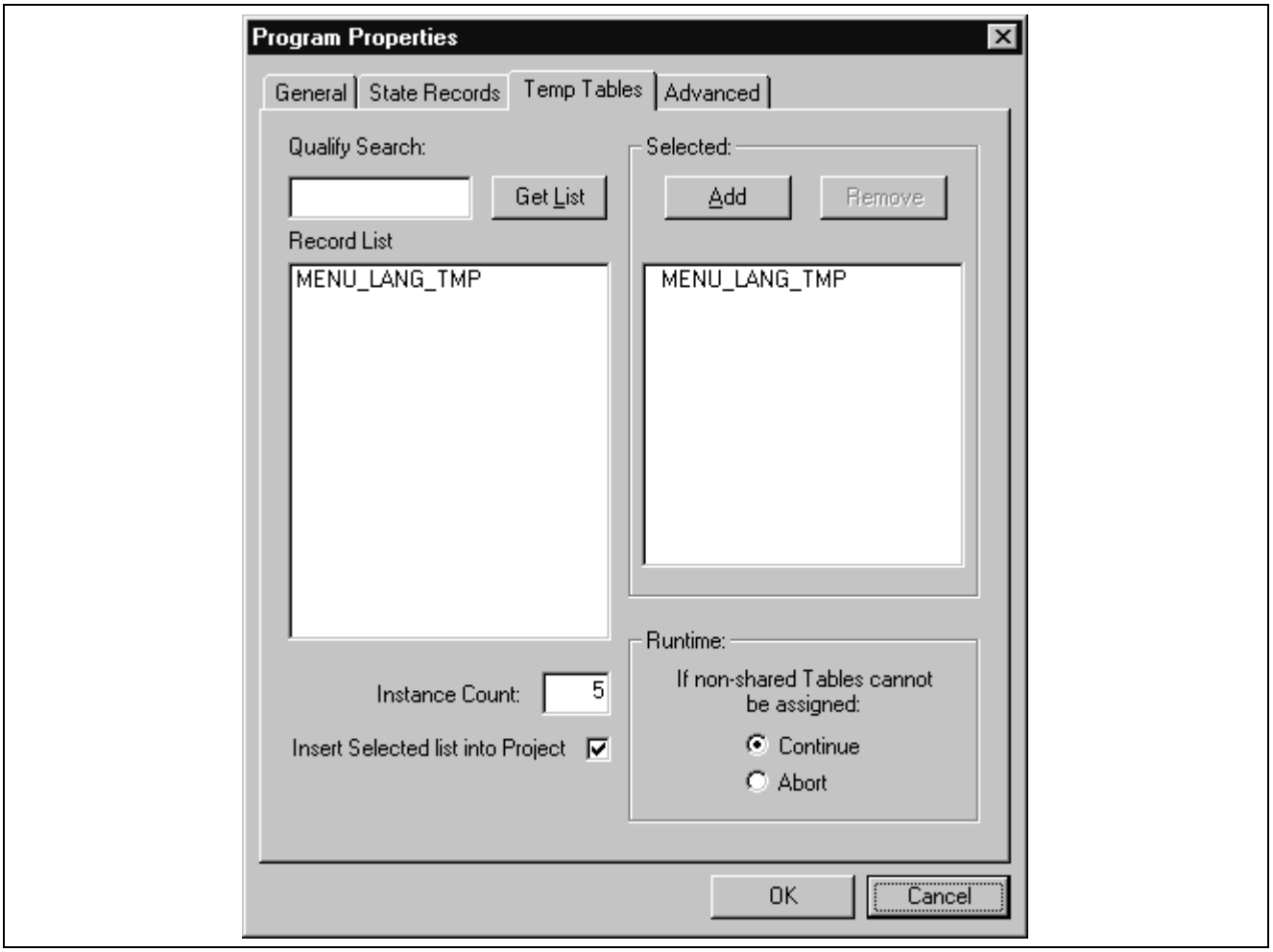
The number of temporary table instances built for a specific temporary table record during the SQL Build process is the value of the total temporary table instances from the PeopleTools Options page added to the sum of all the instance count values specified on the Temp Table tab for the Application Engine programs that use that temporary table.

For example, assume that we have defined APPLTMPA as a temporary record type. If the number of total temporary table instances is set to 10, and APPLTMPA appears in the Temp Tables tab in the Program Properties dialog box for two Application Engine programs. In one program, the instance count is set to 3, and in the other, the instance count is set to 2. When you run the SQL Build process, PeopleTools builds a total of 15 temporary table instances for APPLTMPA.

The total and online instance counts should be equal, unless your PeopleSoft application documentation provides specific instructions on setting these values differently. When the values are equal, the Temp Table Instances (Total) field controls the number of physical temporary table instances to be used by online programs that PeopleSoft Application Designer creates for a temporary table record definition. If the value for the Temp Table Instances (Online) field is less than the value for the Temp Table Instances (Total) field, the difference between the two numbers provides a pool of tables for backward compatibility for developers who took advantage of the %Table(record_name, instance_number) approach for manually managing temporary table locking, (such as PeopleSoft EPM).

Assigning Temporary Tables to Programs

Open an Application Engine program in PeopleSoft Application Designer. Select File, Definition Properties. Select the Temp Tables tab.



Program Properties dialog box: Temp Tables tab

In the Record List box, include all the necessary temporary table records for this program.

In the Instance Count field, specify the number of copies of temporary tables for a program. Anytime you change the instance counts, you need to rebuild the temporary tables to ensure that the right number of instances get created and are available for your programs.

Note. The concept of dedicated temporary tables is isolated to the Application Engine program run. The locking, truncate/delete from, and unlocking are designed to occur within the bounds of an Application Engine program run. Therefore, the system does not keep a temporary table instance available after the Application Engine program run is over.

Runtime Allocation of Temporary Tables

Online processes have their own set of dedicated temporary tables, defined globally on the PeopleTools Options page. When you invoke a process online, PeopleTools randomly allocates a single temporary table instance number to a program for all its dedicated temporary table needs. While the program runs, no other program can use that instance number. Any other online process that happens to get the same instance value waits for the first program to finish, so that the instance number is unlocked.

In contrast, batch processes are allocated temporary table instances on a record-by-record basis. The system begins with the lowest instance number available for each temporary table until all of the temporary table instances are in use. If there are not any temporary tables, available and you selected Continue for the If non-shared Tables cannot be assigned group box, then the base table is used, with the process instance number as a key.

When a program ends normally or is cancelled with Process Monitor, the system automatically releases the assigned instances.

Condition	Online	Batch
Temporary tables are allocated using meta-SQL.	%Table(temp-tbl)	%Table(temp-tbl)
Temporary tables are allocated at runtime.	Psae.exe randomly assigns an instance number from the number range on your online temporary table setting on the PeopleTools Options page. Psae.exe uses that number for all tables for that program run.	Individually allocates an instance number based on availability on a record-by-record basis. Psae.exe begins with the lowest instance number available for each temporary table, until all of the instances are in use.
No temporary tables are free.	For a particular record, if the instance is currently in use, and the program is set to *Continue*, then the psae.exe queues the program until the assigned instance number becomes free.	If the program is set to *Continue,* the system uses a shared base table. If the program is set to *Abort,* then the system terminates the program. *Never* queues for a table.
A temporary table is initially clear.	Yes, when program instance becomes comes available.	Yes, when assigned.

Condition	Online	Batch
An instance number is locked.	The lock is on when the program is loading into memory.	The lock is on when the program is loading into memory. For restartable programs, the temporary tables remain locked across restarts until the program has completed successfully or until the temporary tables are manually released using Process Monitor or the Manage Abends page.
An instance number is unlocked.	Temp tables unlocked on completion of program. In the event of a kill or a crash, the tables remain locked, and the tables must be freed using Process Monitor or the Manage Abends page.	If restart is disabled, the temporary tables are unassigned automatically in the event of a controlled abnormal termination. If you cancel a process using Process Monitor, PeopleTools frees the temporary tables automatically. When you use the Manage Abends page, you must click the Temp Tables button corresponding to the correct process instance, and then click the Release button on the Temporary Tables tab of the Application Engine program properties.

Note. When you have manually released the temporary tables from their locked state, you lose any option to restart the program run.

Sharing Temporary Table Data

Dedicated temporary tables do not remain locked across process instances. If sequential Application Engine programs need to share data by way of temporary tables, a parent Application Engine program should call the programs that share data.

Adjusting Meta-SQL

A critical step in implementing parallel processing is to make sure that you've included appropriate meta-SQL within the code that your Application Engine program executes.

Referencing Temporary Tables

To reference a dedicated temporary table, you must use

```
%Table(record)
```

You can reference any table with %Table, but only those records defined as temporary tables get replaced with a dedicated instance table by PeopleSoft Application Engine. When you are developing programs that take advantage of %Table, choose temporary table indexes carefully. Depending on the use of the temporary table in your program and your data profile, the system indexes may be sufficient. On the other hand, a custom index may be needed instead, or perhaps no indexes are necessary at all. Consider these issues when designing your application. You want to define indexes and SQL that perform well in most situations, but individual programs or environments may require additional performance tuning during implementation.

Note. The default table name refers to PS_recname, where PS_recname1,2,… represents the dedicated temporary tables.

As PeopleSoft Application Engine resolves any %Table, it checks an internal array to see if a temporary table instance has already been chosen for the current record. If so, then PeopleSoft Application Engine substitutes the chosen table name. If not, as in when a record does not appear in the temp table list for the program, then PeopleSoft Application Engine uses the base table instance (PS_recname) by default. Regardless of whether %Table is in PeopleCode SQL or in an Application Engine SQL Action the program uses the same physical SQL table.

Populating the Temporary Table Process Instance with the Process Instance

All temporary tables should be keyed by process instance. If you use the Continue option when batch or dedicated tables can't be assigned, Process Instance is required as a key field. The current process instance is automatically put into the state record, but when you insert rows into your temporary tables, you must supply that process instance. Use %ProcessInstance or %Bind(PROCESS_INSTANCE) meta-SQL to return the numeric (unquoted) process instance.

The process instance value is always zero for programs initiated with the CallAppEngine function. his is because the program called with CallAppEngine runs in process; that is, it runs within the same unit of work as the component with which it is associated.

If you are using dedicated tables and have elected to continue if dedicated tables can't be assigned, then SQL references to dedicated temporary tables must include PROCESS_INSTANCE in the Where clause.

Clearing Temporary Tables

You do not need to delete data from a temporary table manually. The temporary tables are truncated automatically when they are assigned to your program. If the shared base table has been allocated, because no dedicated instances were available, then PeopleSoft Application Engine performs a delete by process instance instead of performing a truncate. In such a case, PROCESS_INSTANCE is required as a high-level key.

You can perform additional deletes of temporary table results during the run, but you must include your own SQL action that uses the %TruncateTable function. If the shared base table has been allocated because no dedicated instances were available, then %TruncateTable is replaced with a delete by process instance instead of a truncate.

Note. You should always use %TruncateTable to perform a mass delete on dedicated temporary tables, especially if the Continue option is in effect.

Even if you have elected to terminate the program if a dedicated table cannot be allocated, you may still use %TruncateTable meta-SQL with dedicated temporary tables. %TruncateTable resolves to either a Truncate or a Delete by process instance, as needed.

The argument of %TruncateTable is a table name instead of a record name. As a result, you must code your SQL as shown in the following example:

```
%TruncateTable(%Table(recname))
```

Note. You should avoid hard-coded table names inside %TruncateTable, since they preclude the possibility of concurrent processing.

Making External Calls

When you call one Application Engine program from another, the assignment of dedicated tables for the called, or child, program, occurs only if the calling, or parent, program is in a state where a commit can occur immediately.

PeopleTools enables you to commit immediately, so that PeopleSoft Application Engine can commit the update it performs to lock the temporary table instance. Otherwise, no other parallel process could perform any assignments. In general, this means that you should issue a commit just prior to the Call Section action.

While making external program calls, note the following:

- If the situation is suitable for a commit, then the temporary table assignment and the appropriate truncates occur.
- If the situation is not suitable for a commit, and the called program is set to continue if dedicated tables cannot be allocated, then the base tables are used instead, and a delete by process instance is performed.
- If the situation is not suitable for a commit and the called program is set to terminate if dedicated tables cannot be allocated, then program execution terminates.

  This reflects an implementation flaw that you need to correct.
- If the called Application Engine program shares temporary tables with the calling program, this is allowed.

  Common temporary tables are the way you share data between the calling and called programs. PeopleSoft Application Engine locks only instances of temporary tables that have not already been used during the current program run. Temporary tables that already have an assigned instance continue to use that instance.

External Calls in Batch Mode

For batch runs, list in the program properties of the root program all of the temporary tables that any called programs or sections use. This ensures that the tables get locked sooner and as a single unit. This approach can improve performance, and it ensures that all the tables required by the program are ready before execution starts.

External Calls in Online Mode

If the online program run is designed to use any temporary tables at any point during the CallAppEngine unit of work, then the root program must have at least one temporary table specified in the Application Engine program properties. This is true even if the root program doesn't use temporary tables. This is required so that the system locks the instance number early on to avoid an instance assignment failure after the process has already started processing.

All temporary tables used by a specific program, library, or external section must be specified in that program to ensure that the system issues truncates (deletes) for the tables being utilized.

If no temporary tables appear in the root program properties, and PeopleSoft Application Engine encounters a %Table reference for a temporary table record, an error appears.

Sample Implementation

The following scenario describes the runtime behavior of PeopleSoft Application Engine and temporary tables.

Assume you have Program A and Program B, and three temporary table definitions: PS_TMPA, PS_TMPB, and PS_TMPC. Values on the Temporary Tables tab in the Program Properties dialog box for each program are as follows:

- Program A: PS_TMPA and PS_TMPB are specified as the dedicated temporary tables, and the instance count is 4.
- Program B: PS_TMPB and PS_TMPC are specified as the dedicated temporary tables, and the instance count is 3.

After you run the SQL Build process in PeopleSoft Application Designer, the following inventory of temporary tables appears in the database.

For PS_TMPA:

- PS_TMPA1
- PS_TMPA2
- PS_TMPA3
- PS_TMPA4

For PS_TMPB:

- PS_TMPB1
- PS_TMPB2
- PS_TMPB3
- PS_TMPB4
- PS_TMPB5
- PS_TMPB6
- PS_TMPB7

For PS_TMPC:

- PS_TMPC1
- PS_TMPC2
- PS_TMPC3

Because the instance count for Program A is 4, the system builds four instances of PS_TMPA and PS_TMPB for Program A to use. Because the instance count for Program B is 3, the system builds an additional three instances of PS_TMPB and three instances of PS_TMPC for Program B to use.

Notice that because Program A and Program B are sharing PS_TMPB, there are seven instances. The system derives this total by adding the instance count value from all the programs that share a particular temporary table instance. In this case, the four from Program A and the three from Program B combine to require a total of seven instances of PS_TMPB to be built.

Given that this collection of temporary tables exists in your database, let’s say that you start Program A. At runtime, PeopleSoft Application Engine examines the list of temporary tables dedicated to Program A, and assigns the first available instances to Program A. Then, assuming that no other programs are running, PeopleSoft Application Engine assigns PS_TMPA1 and PS_TMPB1 to Program A.

Now suppose that shortly after you started Program A, another user starts Program B. Again, PeopleSoft Application Engine examines the list of temporary tables dedicated to Program B and assigns the first available instances. In this scenario, PeopleSoft Application Engine assigns PS_TMPB2 and PS_TMPC1 to Program B. Because Program A is already using PS_TMPB1, the system assigns PS_TMPB2 to Program B.

The system assigns records, such as TMPA, to programs. The base tables, such as PS_TMPA, are also built, by default, in addition to the dedicated temporary instances. If the Program Properties dialog box setting for the Temp Tables tab is set to Continue when no instances are available, the system uses the base table instead of the dedicated instance.

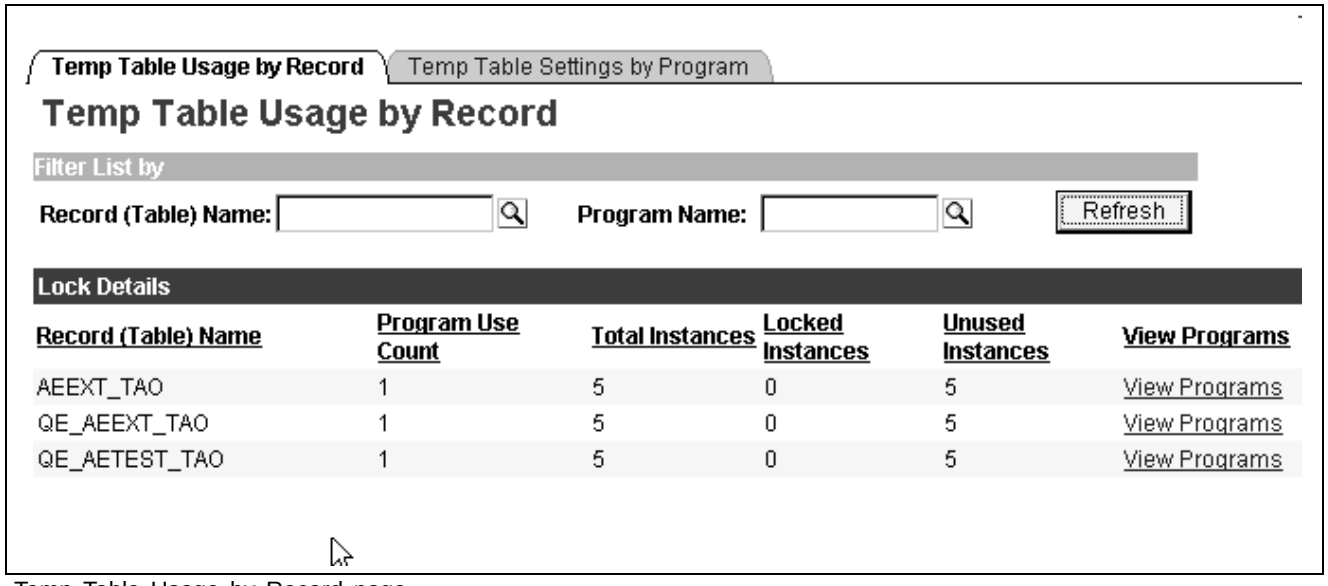
Viewing Temporary Table Usage

This section discusses how to:

- View temporary table usage by record.
- View temporary table settings by program.
- View online instance usage.
- Resolve the temporary table usage warning message.

Viewing Temporary Table Usage by Record

Select PeopleTools, Application Engine, Review Temp Table Usage to access the Temp Table Usage by Record page.



Temp Table Usage by Record page

If you have implemented temporary tables for parallel Application Engine program runs, use this page and the Temp Table Usage by Program page to find out how the system allocates temporary tables to your programs.

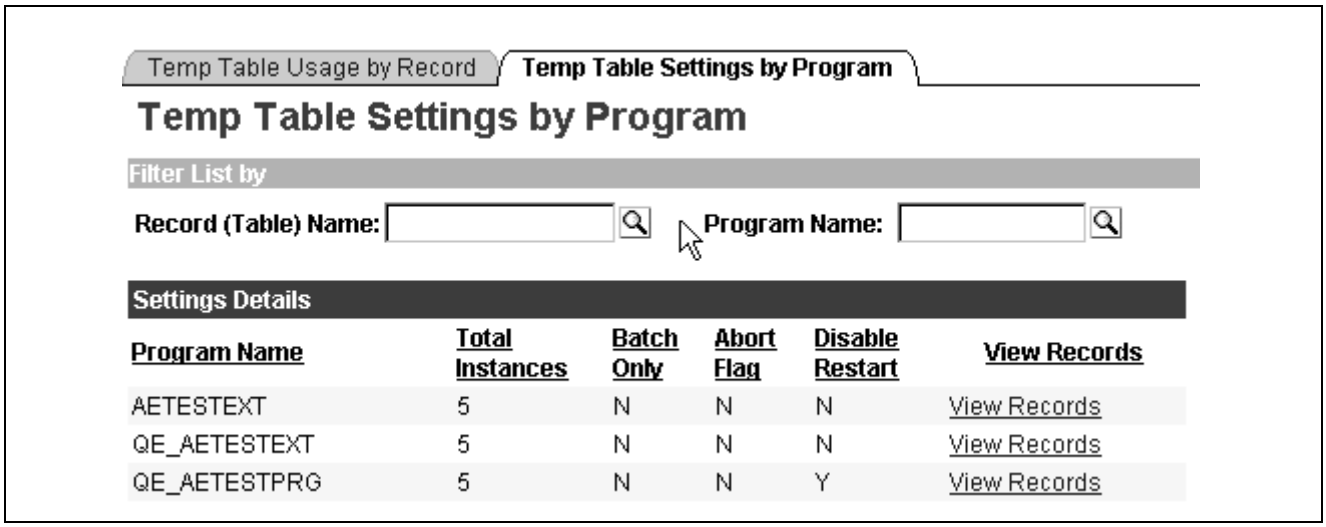
Parallel processing is designed to be a performance enhancing option. However, if the demand for temporary table instances consistently exceeds the current supply, performance suffers. Also, in other situations, your inventory of temporary table instances may far outnumber demand. Here, you may consider reducing the number of instances provided to conserve system resources.

This page shows you the following metrics for evaluating inventory and allocation of temporary tables.

Program Use Count	Shows the instance count of listed program.
Total Instances	Shows the total number of existing instances of a temporary table.
Locked Instances	Shows the current number of instances that they system has locked for program runs.
Unused Instances	Shows the current number of instances that are available for use.

Viewing Temporary Table Settings by Program

Select PeopleTools, Application Engine, Review Temporary Table Usage, Temp Table Settings by Program to access the Temp Table Settings by Program page.

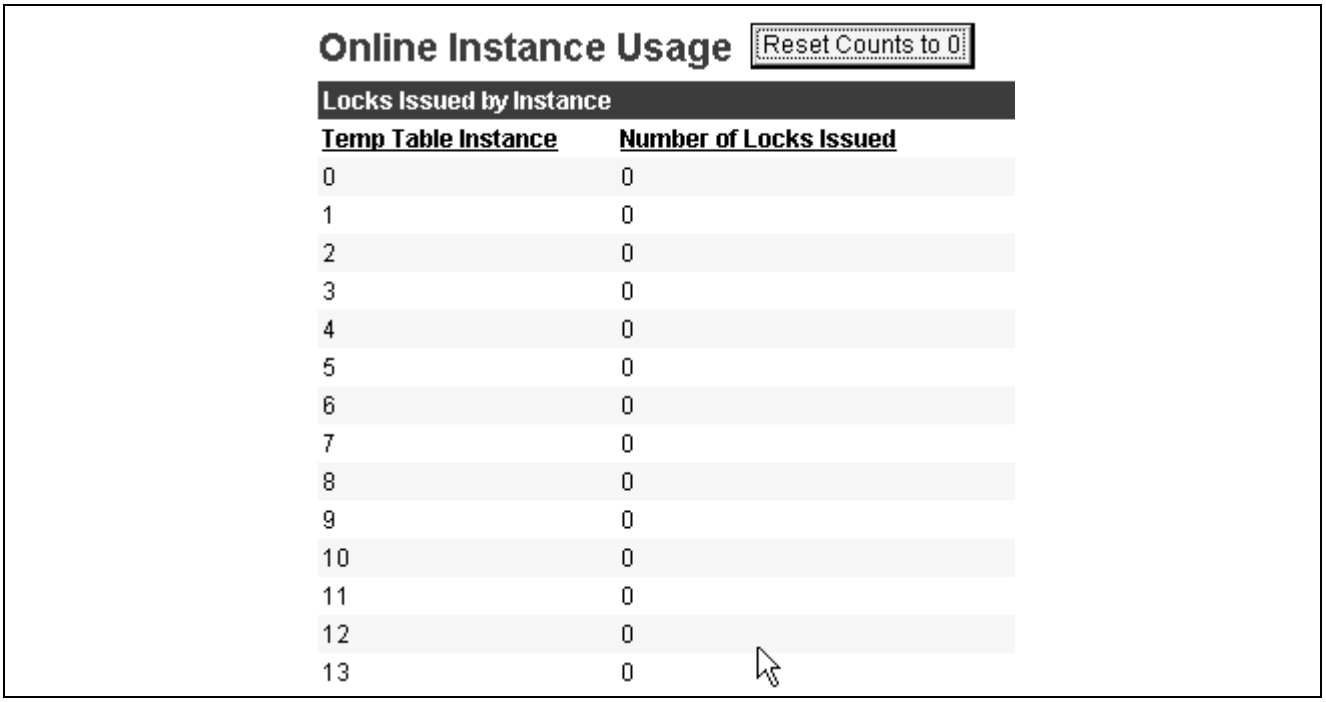


Temp Table Settings by Program page

If the Application Engine process was started in Process Monitor, you can select PeopleTools, Application Engine, Manage Abends to access the Manage Abends page and then the Process Monitor.

Viewing Online Instance Usage

Select PeopleTools, Application Engine, Review Online Instance Usage.



Online Instance Usage page

Resolving the Temporary Table Usage Warning Message

If an Application Engine batch program is unable to get a dedicated temporary table because all instances are locked, but it can use the base table, the system issues a warning. However, if the program has been set to terminate when a dedicated instance is not available, then the program terminates even if the base table can be used.

You could see the warning message in two ways:

- A warning message appears in the standard output of the process.

  When running from the command prompt, the message appears in that window. When the program is running on a server through PeopleSoft Process Scheduler, the output is sent to the standard status file, which you can access using Process Monitor.

- A warning message appears in the AET trace file if a dedicated temporary table instance can't be locked because none are available.

  This message appears in the trace file regardless of the trace settings you've selected.

If you see the warning regarding base temporary table usage, this means either there aren't enough temporary table instances defined or some locked instances that must be released.

When a restartable process terminates abnormally, the temporary tables stay locked to enable a smooth restart. However, if you don't want to restart the process, then the locked temporary tables must be released. When you cancel the process using Process Monitor, the release of locked temporary tables occurs automatically. If the process wasn't launched through PeopleSoft Process Scheduler, Process Monitor does not track the process. Because of this, you must use the Manage Abends page to release temporary tables used by processes invoked outside of PeopleSoft Process Scheduler.

APPENDIX A

ISO Country and Currency Codes

PeopleBooks use International Organization for Standardization (ISO) country and currency codes to identify country-specific information and monetary amounts.

This appendix discusses:

- ISO country codes.
- ISO currency codes.

See Also

"About This PeopleBook." Typographical Conventions and Visual Cues

ISO Country Codes

This table lists the ISO country codes that may appear as country identifiers in PeopleBooks:

ISO Country Code	Country Name
ABW	Aruba
AFG	Afghanistan
AGO	Angola
AIA	Anguilla
ALB	Albania
AND	Andorra
ANT	Netherlands Antilles
ARE	United Arab Emirates
ARG	Argentina
ARM	Armenia
ASM	American Samoa
ATA	Antarctica

ISO Country Code	Country Name
ATF	French Southern Territories
ATG	Antigua and Barbuda
AUS	Australia
AUT	Austria
AZE	Azerbaijan
BDI	Burundi
BEL	Belgium
BEN	Benin
BFA	Burkina Faso
BGD	Bangladesh
BGR	Bulgaria
BHR	Bahrain
BHS	Bahamas
BIH	Bosnia and Herzegovina
BLR	Belarus
BLZ	Belize
BMU	Bermuda
BOL	Bolivia
BRA	Brazil
BRB	Barbados
BRN	Brunei Darussalam
BTN	Bhutan
BVT	Bouvet Island
BWA	Botswana
CAF	Central African Republic
CAN	Canada
CCK	Cocos (Keeling) Islands

ISO Country Code	Country Name
CHE	Switzerland
CHL	Chile
CHN	China
CIV	Cote D'Ivoire
CMR	Cameroon
COD	Congo, The Democratic Republic
COG	Congo
COK	Cook Islands
COL	Colombia
COM	Comoros
CPV	Cape Verde
CRI	Costa Rica
CUB	Cuba
CXR	Christmas Island
CYM	Cayman Islands
CYP	Cyprus
CZE	Czech Republic
DEU	Germany
DJI	Djibouti
DMA	Dominica
DNK	Denmark
DOM	Dominican Republic
DZA	Algeria
ECU	Ecuador
EGY	Egypt
ERI	Eritrea
ESH	Western Sahara

ISO Country Code	Country Name
ESP	Spain
EST	Estonia
ETH	Ethiopia
FIN	Finland
FJI	Fiji
FLK	Falkland Islands (Malvinas)
FRA	France
FRO	Faroe Islands
FSM	Micronesia, Federated States
GAB	Gabon
GBR	United Kingdom
GEO	Georgia
GHA	Ghana
GIB	Gibraltar
GIN	Guinea
GLP	Guadeloupe
GMB	Gambia
GNB	Guinea-Bissau
GNQ	Equatorial Guinea
GRC	Greece
GRD	Grenada
GRL	Greenland
GTM	Guatemala
GUF	French Guiana
GUM	Guam
GUY	Guyana
GXA	GXA - GP Core Country

ISO Country Code	Country Name
GXB	GXB - GP Core Country
GXC	GXC - GP Core Country
GXD	GXD - GP Core Country
HKG	Hong Kong
HMD	Heard and McDonald Islands
HND	Honduras
HRV	Croatia
HTI	Haiti
HUN	Hungary
IDN	Indonesia
IND	India
IOT	British Indian Ocean Territory
IRL	Ireland
IRN	Iran (Islamic Republic Of)
IRQ	Iraq
ISL	Iceland
ISR	Israel
ITA	Italy
JAM	Jamaica
JOR	Jordan
JPN	Japan
KAZ	Kazakstan
KEN	Kenya
KGZ	Kyrgyzstan
KHM	Cambodia
KIR	Kiribati
KNA	Saint Kitts and Nevis

ISO Country Code	Country Name
KOR	Korea, Republic of
KWT	Kuwait
LAO	Lao People’s Democratic Rep
LBN	Lebanon
LBR	Liberia
LBY	Libyan Arab Jamahiriya
LCA	Saint Lucia
LIE	Liechtenstein
LKA	Sri Lanka
LSO	Lesotho
LTU	Lithuania
LUX	Luxembourg
LVA	Latvia
MAC	Macao
MAR	Morocco
MCO	Monaco
MDA	Moldova, Republic of
MDG	Madagascar
MDV	Maldives
MEX	Mexico
MHL	Marshall Islands
MKD	Fmr Yugoslav Rep of Macedonia
MLI	Mali
MLT	Malta
MMR	Myanmar
MNG	Mongolia
MNP	Northern Mariana Islands

ISO Country Code	Country Name
MOZ	Mozambique
MRT	Mauritania
MSR	Montserrat
MTQ	Martinique
MUS	Mauritius
MWI	Malawi
MYS	Malaysia
MYT	Mayotte
NAM	Namibia
NCL	New Caledonia
NER	Niger
NFK	Norfolk Island
NGA	Nigeria
NIC	Nicaragua
NIU	Niue
NLD	Netherlands
NOR	Norway
NPL	Nepal
NRU	Nauru
NZL	New Zealand
OMN	Oman
PAK	Pakistan
PAN	Panama
PCN	Pitcairn
PER	Peru
PHL	Philippines
PLW	Palau

ISO Country Code	Country Name
PNG	Papua New Guinea
POL	Poland
PRI	Puerto Rico
PRK	Korea, Democratic People’s Rep
PRT	Portugal
PRY	Paraguay
PSE	Palestinian Territory, Occupie
PYF	French Polynesia
QAT	Qatar
REU	Reunion
ROU	Romania
RUS	Russian Federation
RWA	Rwanda
SAU	Saudi Arabia
SDN	Sudan
SEN	Senegal
SGP	Singapore
SGS	Sth Georgia & Sth Sandwich Is
SHN	Saint Helena
SJM	Svalbard and Jan Mayen
SLB	Solomon Islands
SLE	Sierra Leone
SLV	El Salvador
SMR	San Marino
SOM	Somalia
SPM	Saint Pierre and Miquelon
STP	Sao Tome and Principe

ISO Country Code	Country Name
SUR	Suriname
SVK	Slovakia
SVN	Slovenia
SWE	Sweden
SWZ	Swaziland
SYC	Seychelles
SYR	Syrian Arab Republic
TCA	Turks and Caicos Islands
TCD	Chad
TGO	Togo
THA	Thailand
TJK	Tajikistan
TKL	Tokelau
TKM	Turkmenistan
TLS	East Timor
TON	Tonga
TTO	Trinidad and Tobago
TUN	Tunisia
TUR	Turkey
TUV	Tuvalu
TWN	Taiwan, Province of China
TZA	Tanzania, United Republic of
UGA	Uganda
UKR	Ukraine
UMI	US Minor Outlying Islands
URY	Uruguay
USA	United States

ISO Country Code	Country Name
UZB	Uzbekistan
VAT	Holy See (Vatican City State)
VCT	St Vincent and the Grenadines
VEN	Venezuela
VGB	Virgin Islands (British)
VIR	Virgin Islands (U.S.)
VNM	Viet Nam
VUT	Vanuatu
WLF	Wallis and Futuna Islands
WSM	Samoa
YEM	Yemen
YUG	Yugoslavia
ZAF	South Africa
ZMB	Zambia
ZWE	Zimbabwe

ISO Currency Codes

This table lists the ISO country codes that may appear as currency identifiers in PeopleBooks:

ISO Currency Code	Description
ADP	Andorran Peseta
AED	United Arab Emirates Dirham
AFA	Afghani
AFN	Afghani
ALK	Old Lek
ALL	Lek
AMD	Armenian Dram

ISO Currency Code	Description
ANG	Netherlands Antilles Guilder
AOA	Kwanza
AOK	Kwanza
AON	New Kwanza
AOR	Kwanza Reajustado
ARA	Austral
ARP	Peso Argentino
ARS	Argentine Peso
ARY	Peso
ATS	Schilling
AUD	Australian Dollar
AWG	Aruban Guilder
AZM	Azerbaijanian Manat
BAD	Dinar
BAM	Convertible Marks
BBD	Barbados Dollar
BDT	Taka
BEC	Convertible Franc
BEF	Belgian Franc
BEL	Financial Belgian Franc
BGJ	Lev A/52
BGK	Lev A/62
BGL	Lev
BGN	Bulgarian LEV
BHD	Bahraini Dinar
BIF	Burundi Franc
BMD	Bermudian Dollar

ISO Currency Code	Description
BND	Brunei Dollar
BOB	Boliviano
BOP	Peso
BOV	Mvdol
BRB	Cruzeiro
BRC	Cruzado
BRE	Cruzeiro
BRL	Brazilian Real
BRN	New Cruzado
BRR	Brazilian Real Dollar
BSD	Bahamian Dollar
BTN	Ngultrum
BUK	N/A
BWP	Pula
BYB	Belarussian Ruble
BYR	Belarussian Ruble
BZD	Belize Dollar
CAD	Canadian Dollar
CDF	Franc Congolais
CHF	Swiss Franc
CLF	Unidades de fomento
CLP	Chilean Peso
CNX	Peoples Bank Dollar
CNY	Yuan Renminbi
COP	Colombian Peso
CRC	Costa Rican Colon
CSD	Serbia Dinar

ISO Currency Code	Description
CSJ	Krona A/53
CSK	Koruna
CUP	Cuban Peso
CVE	Cape Verde Escudo
CYP	Cyprus Pound
CZK	Czech Koruna
DEM	Deutsche Mark
DJF	Djibouti Franc
DKK	Danish Krone
DOP	Dominican Peso
DZD	Algerian Dinar
ECS	Sucre
ECV	Unidad de Valor
EEK	Kroon
EGP	Egyptian Pound
EQE	Ekwele
ERN	Nakfa
ESA	Spanish Peseta
ESB	Convertible Peseta
ESP	Spanish Peseta
ETB	Ethiopian Birr
EUR	euro
FIM	Markka
FJD	Fiji Dollar
FKP	Falklands Isl. Pound
FRF	French Franc
GBP	Pound Sterling

ISO Currency Code	Description
GEK	Georgian Coupon
GEL	Lari
GHC	Cedi
GIP	Gibraltar Pound
GMD	Dalasi
GNE	Syli
GNF	Guinea Franc
GNS	Syli
GQE	Ekwele
GRD	Drachma
GTQ	Quetzal
GWE	Guinea Escudo
GWP	Guinea-Bissau Peso
GYD	Guyana Dollar
HKD	Hong Kong Dollar
HNL	Lempira
HRD	Dinar
HRK	Kuna
HTG	Gourde
HUF	Forint
IDR	Rupiah
IEP	Irish Pound
ILP	Pound
ILR	Old Shekel
ILS	New Israeli Sheqel
INR	Indian Rupee
IQD	Iraqi Dinar

ISO Currency Code	Description
IRR	Iranian Rial
ISJ	Old Krona
ISK	Iceland Krona
ITL	Italian Lira
JMD	Jamaican Dollar
JOD	Jordanian Dinar
JPY	Yen
KES	Kenyan Shilling
KGS	Som
KHR	Riel
KMF	Comoro Franc
KPW	North Korean Won
KRW	Won
KWD	Kuwaiti Dinar
KYD	Cayman Islands dollar
KZT	Tenge
LAJ	Kip Pot Pol
LAK	Kip
LBP	Lebanese Pound
LKR	Sri Lanka Rupee
LRD	Liberian Dollar
LSL	Loti
LSM	Maloti
LTL	Lithuanian Litas
LTT	Talonas
LUC	Convertib Francl
LUF	Luxembourg Franc

ISO Currency Code	Description
LUL	Financial Franc
LVL	Latvian Lats
LVR	Latvian Ruble
LYD	Libyan Dinar
MAD	Moroccan Dirham
MAF	Mali Franc
MDL	Moldovan Leu
MGF	Malagasy Franc
MKD	Denar
MLF	Mali Franc
MMK	Kyat
MNT	Tugrik
MOP	Pataca
MRO	Ouguiya
MTL	Maltese Lira
MTP	Maltese Pound
MUR	Mauritius Rupee
MVQ	Maldive Rupee
MVR	Rufiyaa
MWK	Malawian Kwacha
MXN	Mexican Peso
MXP	Mexican Peso
MXV	Mexican UDI
MYR	Malaysian Ringgit
MZE	Mozambique Escudo
MZM	Metical
NAD	Namibia Dollar

ISO Currency Code	Description
NGN	Naira
NIC	Cordoba
NIO	Cordoba Oro
NLG	Netherlands Guilder
NOK	Norwegian Krone
NPR	Nepalese Rupee
NZD	New Zealand Dollar
OMR	Rial Omani
PAB	Balboa
PEI	Inti
PEN	Nuevo Sol
PES	Sol
PGK	Kina
PHP	Philippine Peso
PKR	Pakistan Rupee
PLN	Zloty
PLZ	Zloty
PTE	Portuguese Escudo
PYG	Guarani
QAR	Qatari Rial
ROK	Leu A/52
ROL	Leu
RUB	Russian Ruble
RUR	Russian Federation Rouble
RWF	Rwanda Franc
SAR	Saudi Riyal
SBD	Solomon Islands

ISO Currency Code	Description
SCR	Seychelles Rupee
SDD	Sudanese Dinar
SDP	Sudanese Pound
SEK	Swedish Krona
SGD	Singapore Dollar
SHP	St Helena Pound
SIT	Tolar
SKK	Slovak Koruna
SLL	Leone
SOS	Somali Shilling
SRG	Surinam Guilder
STD	Dobra
SUR	Rouble
SVC	El Salvador Colon
SYP	Syrian Pound
SZL	Lilangeni
THB	Baht
TJR	Tajik Ruble
TJS	Somoni
TMM	Manat
TND	Tunisian Dinar
TOP	Pa’anga
TPE	Timor Escudo
TRL	Turkish Lira
TTD	Trinidad Dollar
TWD	New Taiwan Dollar
TZS	Tanzanian Shilling

ISO Currency Code	Description
UAH	Hryvnia
UAK	Karbovanet
UGS	Uganda Shilling
UGW	Old Shilling
UGX	Uganda Shilling
USD	US Dollar
USN	US Dollar (Next day)
USS	US Dollar (Same day)
UYN	Old Uruguay Peso
UYP	Uruguayan Peso
UYU	Peso Uruguayo
UZS	Uzbekistan Sum
VEB	Bolivar
VNC	Old Dong
VND	Dong
VUV	Vatu
WST	Tala
XAF	CFA Franc BEAC
XAG	Silver
XAU	GOLD
XBA	European Composite Unit
XBB	European Monetary Unit
XBC	European Unit of Account 9
XBD	European Unit of Account 17
XCD	East Caribbean Dollar
XDR	SDR
XEU	EU Currency (E.C.U)

ISO Currency Code	Description
XFO	Gold-Franc
XFU	UIC-Franc
XOF	CFA Franc BCEAO
XPD	Palladium
XPF	CFP Franc
XPT	Platinum
XTS	For Testing Purposes
XXX	Non Currency Transaction
YDD	Yemeni Din
YER	Yemeni Rial
YUD	New Yugoslavian Dinar
YUM	New Dinar
YUN	Yugoslavian Dinar
ZAL	Financial Rand
ZAR	Rand
ZMK	Zambian Kwacha
ZRN	New Zaire
ZRZ	Zaire
ZWC	Rhodesian Dollar
ZWD	Zimbabwe Dollar

Glossary of PeopleSoft Terms

absence entitlement This element defines rules for granting paid time off for valid absences, such as sick time, vacation, and maternity leave. An absence entitlement element defines the entitlement amount, frequency, and entitlement period.

absence take This element defines the conditions that must be met before a payee is entitled to take paid time off.

accounting class In PeopleSoft Enterprise Performance Management, the accounting class defines how a resource is treated for generally accepted accounting practices. The Inventory class indicates whether a resource becomes part of a balance sheet account, such as inventory or fixed assets, while the Non-inventory class indicates that the resource is treated as an expense of the period during which it occurs.

accounting date The accounting date indicates when a transaction is recognized, as opposed to the date the transaction actually occurred. The accounting date and transaction date can be the same. The accounting date determines the period in the general ledger to which the transaction is to be posted. You can only select an accounting date that falls within an open period in the ledger to which you are posting. The accounting date for an item is normally the invoice date.

accounting split The accounting split method indicates how expenses are allocated or divided among one or more sets of accounting ChartFields.

accumulator You use an accumulator to store cumulative values of defined items as they are processed. You can accumulate a single value over time or multiple values over time. For example, an accumulator could consist of all voluntary deductions, or all company deductions, enabling you to accumulate amounts. It allows total flexibility for time periods and values accumulated.

action reason The reason an employee's job or employment information is updated. The action reason is entered in two parts: a personnel action, such as a promotion, termination, or change from one pay group to another—and a reason for that action. Action reasons are used by PeopleSoft Human Resources, PeopleSoft Benefits Administration, PeopleSoft Stock Administration, and the COBRA Administration feature of the Base Benefits business process.

action template In PeopleSoft Receivables, outlines a set of escalating actions that the system or user performs based on the period of time that a customer or item has been in an action plan for a specific condition.

activity In PeopleSoft Enterprise Learning Management, an instance of a catalog item (sometimes called a class) that is available for enrollment. The activity defines such things as the costs that are associated with the offering, enrollment limits and deadlines, and waitlisting capacities.

In PeopleSoft Enterprise Performance Management, the work of an organization and the aggregation of actions that are used for activity-based costing.

In PeopleSoft Project Costing, the unit of work that provides a further breakdown of projects—usually into specific tasks.

In PeopleSoft Workflow, a specific transaction that you might need to perform in a business process. Because it consists of the steps that are used to perform a transaction, it is also known as a step map.

agreement In PeopleSoft eSettlements, provides a way to group and specify processing options, such as payment terms, pay from a bank, and notifications by a buyer and supplier location combination.

allocation rule In PeopleSoft Enterprise Incentive Management, an expression within compensation plans that enables the system to assign transactions to nodes and participants. During transaction allocation, the allocation engine traverses the compensation structure from the current node to the root node, checking each node for plans that contain allocation rules.

alternate account A feature in PeopleSoft General Ledger that enables you to create a statutory chart of accounts and enter statutory account transactions at the detail transaction level, as required for recording and reporting by some national governments.

AR specialist Abbreviation for *receivables specialist.* In PeopleSoft Receivables, an individual in who tracks and resolves deductions and disputed items.

arbitration plan In PeopleSoft Enterprise Pricer, defines how price rules are to be applied to the base price when the transaction is priced.

assessment rule In PeopleSoft Receivables, a user-defined rule that the system uses to evaluate the condition of a customer's account or of individual items to determine whether to generate a follow-up action.

asset class An asset group used for reporting purposes. It can be used in conjunction with the asset category to refine asset classification.

attribute/value pair In PeopleSoft Directory Interface, relates the data that makes up an entry in the directory information tree.

authentication server A server that is set up to verify users of the system.

base time period In PeopleSoft Business Planning, the lowest level time period in a calendar.

benchmark job In PeopleSoft Workforce Analytics, a benchmark job is a job code for which there is corresponding salary survey data from published, third-party sources.

book In PeopleSoft Asset Management, used for storing financial and tax information, such as costs, depreciation attributes, and retirement information on assets.

branch A tree node that rolls up to nodes above it in the hierarchy, as defined in PeopleSoft Tree Manager.

budgetary account only An account used by the system only and not by users; this type of account does not accept transactions. You can only budget with this account. Formerly called "system-maintained account."

budget check In commitment control, the processing of source transactions against control budget ledgers, to see if they pass, fail, or pass with a warning.

budget control In commitment control, budget control ensures that commitments and expenditures don't exceed budgets. It enables you to track transactions against corresponding budgets and terminate a document's cycle if the defined budget conditions are not met. For example, you can prevent a purchase order from being dispatched to a vendor if there are insufficient funds in the related budget to support it.

budget period The interval of time (such as 12 months or 4 quarters) into which a period is divided for budgetary and reporting purposes. The ChartField allows maximum flexibility to define operational accounting time periods without restriction to only one calendar.

business event In PeopleSoft Receivables, defines the processing characteristics for the Receivable Update process for a draft activity.

	In PeopleSoft Sales Incentive Management, an original business transaction or activity that may justify the creation of a PeopleSoft Enterprise Incentive Management event (a sale, for example).
business unit	A corporation or a subset of a corporation that is independent with regard to one or more operational or accounting functions.
buyer	In PeopleSoft eSettlements, an organization (or business unit, as opposed to an individual) that transacts with suppliers (vendors) within the system. A buyer creates payments for purchases that are made in the system.
catalog item	In PeopleSoft Enterprise Learning Management, a specific topic that a learner can study and have tracked. For example, “Introduction to Microsoft Word.” A catalog item contains general information about the topic and includes a course code, description, categorization, keywords, and delivery methods. A catalog item can have one or more learning activities.
catalog map	In PeopleSoft Catalog Management, translates values from the catalog source data to the format of the company’s catalog.
catalog partner	In PeopleSoft Catalog Management, shares responsibility with the enterprise catalog manager for maintaining catalog content.
categorization	Associates partner offerings with catalog offerings and groups them into enterprise catalog categories.
channel	In PeopleSoft MultiChannel Framework, email, chat, voice (computer telephone integration [CTI]), or a generic event.
ChartField	A field that stores a chart of accounts, resources, and so on, depending on the PeopleSoft application. ChartField values represent individual account numbers, department codes, and so forth.
ChartField balancing	You can require specific ChartFields to match up (balance) on the debit and the credit side of a transaction.
ChartField combination edit	The process of editing journal lines for valid ChartField combinations based on user-defined rules.
ChartKey	One or more fields that uniquely identify each row in a table. Some tables contain only one field as the key, while others require a combination.
checkbook	In PeopleSoft Promotions Management, enables you to view financial data (such as planned, incurred, and actual amounts) that is related to funds and trade promotions.
Class ChartField	A ChartField value that identifies a unique appropriation budget key when you combine it with a fund, department ID, and program code, as well as a budget period. Formerly called *sub-classification.*
clone	In PeopleCode, to make a unique copy. In contrast, to *copy* may mean making a new reference to an object, so if the underlying object is changed, both the copy and the original change.
collection	To make a set of documents available for searching in Verity, you must first create at least one collection. A collection is set of directories and files that allow search application users to use the Verity search engine to quickly find and display source documents that match search criteria. A collection is a set of statistics and pointers to the source documents, stored in a proprietary format on a file server. Because a collection can only store information for a single location, PeopleSoft maintains a set of collections (one per language code) for each search index object.

collection rule In PeopleSoft Receivables, a user-defined rule that defines actions to take for a customer based on both the amount and the number of days past due for outstanding balances.

compensation object In PeopleSoft Enterprise Incentive Management, a node within a compensation structure. Compensation objects are the building blocks that make up a compensation structure's hierarchical representation.

compensation structure In PeopleSoft Enterprise Incentive Management, a hierarchical relationship of compensation objects that represents the compensation-related relationship between the objects.

condition In PeopleSoft Receivables, occurs when there is a change of status for a customer's account, such as reaching a credit limit or exceeding a user-defined balance due.

configuration parameter catalog Used to configure an external system with PeopleSoft. For example, a configuration parameter catalog might set up configuration and communication parameters for an external server.

configuration plan In PeopleSoft Enterprise Incentive Management, configuration plans hold allocation information for common variables (not incentive rules) and are attached to a node without a participant. Configuration plans are not processed by transactions.

content reference Content references are pointers to content registered in the portal registry. These are typically either URLs or iScripts. Content references fall into three categories: target content, templates, and template pagelets.

context In PeopleCode, determines which buffer fields can be contextually referenced and which is the current row of data on each scroll level when a PeopleCode program is running.

In PeopleSoft Enterprise Incentive Management, a mechanism that is used to determine the scope of a processing run. PeopleSoft Enterprise Incentive Management uses three types of context: plan, period, and run-level.

control table Stores information that controls the processing of an application. This type of processing might be consistent throughout an organization, or it might be used only by portions of the organization for more limited sharing of data.

cost profile A combination of a receipt cost method, a cost flow, and a deplete cost method. A profile is associated with a cost book and determines how items in that book are valued, as well as how the material movement of the item is valued for the book.

cost row A cost transaction and amount for a set of ChartFields.

current learning In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's in-progress learning activities and programs.

data acquisition In PeopleSoft Enterprise Incentive Management, the process during which raw business transactions are acquired from external source systems and fed into the operational data store (ODS).

data elements Data elements, at their simplest level, define a subset of data and the rules by which to group them.

For Workforce Analytics, data elements are rules that tell the system what measures to retrieve about your workforce groups.

dataset A data grouping that enables role-based filtering and distribution of data. You can limit the range and quantity of data that is displayed for a user by associating dataset rules with user roles. The result of dataset rules is a set of data that is appropriate for the user's roles.

delivery method In PeopleSoft Enterprise Learning Management, identifies the primary type of delivery method in which a particular learning activity is offered. Also provides default values for the learning activity, such as cost and language. This is primarily used to help learners search the catalog for the type of delivery from which they learn best. Because PeopleSoft Enterprise Learning Management is a blended learning system, it does not enforce the delivery method.

In PeopleSoft Supply Chain Management, identifies the method by which goods are shipped to their destinations (such as truck, air, rail, and so on). The delivery method is specified when creating shipment schedules.

delivery method type In PeopleSoft Enterprise Learning Management, identifies how learning activities can be delivered—for example, through online learning, classroom instruction, seminars, books, and so forth—in an organization. The type determines whether the delivery method includes scheduled components.

directory information tree In PeopleSoft Directory Interface, the representation of a directory's hierarchical structure.

document sequencing A flexible method that sequentially numbers the financial transactions (for example, bills, purchase orders, invoices, and payments) in the system for statutory reporting and for tracking commercial transaction activity.

dynamic detail tree A tree that takes its detail values—dynamic details—directly from a table in the database, rather than from a range of values that are entered by the user.

edit table A table in the database that has its own record definition, such as the Department table. As fields are entered into a PeopleSoft application, they can be validated against an edit table to ensure data integrity throughout the system.

effective date A method of dating information in PeopleSoft applications. You can predate information to add historical data to your system, or postdate information in order to enter it before it actually goes into effect. By using effective dates, you don't delete values; you enter a new value with a current effective date.

EIM ledger Abbreviation for *Enterprise Incentive Management ledger.* In PeopleSoft Enterprise Incentive Management, an object to handle incremental result gathering within the scope of a participant. The ledger captures a result set with all of the appropriate traces to the data origin and to the processing steps of which it is a result.

elimination set In PeopleSoft General Ledger, a related group of intercompany accounts that is processed during consolidations.

entry event In PeopleSoft General Ledger, Receivables, Payables, Purchasing, and Billing, a business process that generates multiple debits and credits resulting from single transactions to produce standard, supplemental accounting entries.

equitization In PeopleSoft General Ledger, a business process that enables parent companies to calculate the net income of subsidiaries on a monthly basis and adjust that amount to increase the investment amount and equity income amount before performing consolidations.

event A predefined point either in the Component Processor flow or in the program flow. As each point is encountered, the event activates each component, triggering any PeopleCode program that is associated with that component and that event. Examples of events are FieldChange, SavePreChange, and RowDelete.

In PeopleSoft Human Resources, also refers to an incident that affects benefits eligibility.

event propagation process In PeopleSoft Sales Incentive Management, a process that determines, through logic, the propagation of an original PeopleSoft Enterprise Incentive Management event and creates a derivative (duplicate) of the original event to be processed by other objects.

	Sales Incentive Management uses this mechanism to implement splits, roll-ups, and so on. Event propagation determines who receives the credit.
exception	In PeopleSoft Receivables, an item that either is a deduction or is in dispute.
exclusive pricing	In PeopleSoft Order Management, a type of arbitration plan that is associated with a price rule. Exclusive pricing is used to price sales order transactions.
fact	In PeopleSoft applications, facts are numeric data values from fields from a source database as well as an analytic application. A fact can be anything you want to measure your business by, for example, revenue, actual, budget data, or sales numbers. A fact is stored on a fact table.
forecast item	A logical entity with a unique set of descriptive demand and forecast data that is used as the basis to forecast demand. You create forecast items for a wide range of uses, but they ultimately represent things that you buy, sell, or use in your organization and for which you require a predictable usage.
fund	In PeopleSoft Promotions Management, a budget that can be used to fund promotional activity. There are four funding methods: top down, fixed accrual, rolling accrual, and zero-based accrual.
generic process type	In PeopleSoft Process Scheduler, process types are identified by a generic process type. For example, the generic process type SQR includes all SQR process types, such as SQR process and SQR report.
group	In PeopleSoft Billing and Receivables, a posting entity that comprises one or more transactions (items, deposits, payments, transfers, matches, or write-offs). In PeopleSoft Human Resources Management and Supply Chain Management, any set of records that are associated under a single name or variable to run calculations in PeopleSoft business processes. In PeopleSoft Time and Labor, for example, employees are placed in groups for time reporting purposes.
incentive object	In PeopleSoft Enterprise Incentive Management, the incentive-related objects that define and support the PeopleSoft Enterprise Incentive Management calculation process and results, such as plan templates, plans, results data, user interaction objects, and so on.
incentive rule	In PeopleSoft Sales Incentive Management, the commands that act on transactions and turn them into compensation. A rule is one part in the process of turning a transaction into compensation.
incur	In PeopleSoft Promotions Management, to become liable for a promotional payment. In other words, you owe that amount to a customer for promotional activities.
item	In PeopleSoft Inventory, a tangible commodity that is stored in a business unit (shipped from a warehouse). In PeopleSoft Demand Planning, Inventory Policy Planning, and Supply Planning, a noninventory item that is designated as being used for planning purposes only. It can represent a family or group of inventory items. It can have a planning bill of material (BOM) or planning routing, and it can exist as a component on a planning BOM. A planning item cannot be specified on a production or engineering BOM or routing, and it cannot be used as a component in a production. The quantity on hand will never be maintained. In PeopleSoft Receivables, an individual receivable. An item can be an invoice, a credit memo, a debit memo, a write-off, or an adjustment.
KPI	An abbreviation for *key performance indicator.* A high-level measurement of how well an organization is doing in achieving critical success factors. This defines the data value or calculation upon which an assessment is determined.

LDIF file Abbreviation for *Lightweight Directory Access Protocol (LDAP) Data Interchange Format file.* Contains discrepancies between PeopleSoft data and directory data.

learner group In PeopleSoft Enterprise Learning Management, a group of learners who are linked to the same learning environment. Members of the learner group can share the same attributes, such as the same department or job code. Learner groups are used to control access to and enrollment in learning activities and programs. They are also used to perform group enrollments and mass enrollments in the back office.

learning components In PeopleSoft Enterprise Learning Management, the foundational building blocks of learning activities. PeopleSoft Enterprise Learning Management supports six basic types of learning components: web-based, session, webcast, test, survey, and assignment. One or more of these learning component types compose a single learning activity.

learning environment In PeopleSoft Enterprise Learning Management, identifies a set of categories and catalog items that can be made available to learner groups. Also defines the default values that are assigned to the learning activities and programs that are created within a particular learning environment. Learning environments provide a way to partition the catalog so that learners see only those items that are relevant to them.

learning history In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's completed learning activities and programs.

ledger mapping You use ledger mapping to relate expense data from general ledger accounts to resource objects. Multiple ledger line items can be mapped to one or more resource IDs. You can also use ledger mapping to map dollar amounts (referred to as *rates*) to business units. You can map the amounts in two different ways: an actual amount that represents actual costs of the accounting period, or a budgeted amount that can be used to calculate the capacity rates as well as budgeted model results. In PeopleSoft Enterprise Warehouse, you can map general ledger accounts to the EW Ledger table.

library section In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan (or template) and that is available for other plans to share. Changes to a library section are reflected in all plans that use it.

linked section In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan template but appears in a plan. Changes to linked sections propagate to plans using that section.

linked variable In PeopleSoft Enterprise Incentive Management, a variable that is defined and maintained in a plan template and that also appears in a plan. Changes to linked variables propagate to plans using that variable.

load In PeopleSoft Inventory, identifies a group of goods that are shipped together. Load management is a feature of PeopleSoft Inventory that is used to track the weight, the volume, and the destination of a shipment.

local functionality In PeopleSoft HRMS, the set of information that is available for a specific country. You can access this information when you click the appropriate country flag in the global window, or when you access it by a local country menu.

location Locations enable you to indicate the different types of addresses—for a company, for example, one address to receive bills, another for shipping, a third for postal deliveries, and a separate street address. Each address has a different location number. The primary location—indicated by a *1*—is the address you use most often and may be different from the main address.

logistical task In PeopleSoft Services Procurement, an administrative task that is related to hiring a service provider. Logistical tasks are linked to the service type on the work order so that different types of services can have different logistical tasks. Logistical tasks include both preapproval tasks (such as assigning a new badge or ordering a new

	laptop) and postapproval tasks (such as scheduling orientation or setting up the service provider email). The logistical tasks can be mandatory or optional. Mandatory preapproval tasks must be completed before the work order is approved. Mandatory postapproval tasks, on the other hand, must be completed before a work order is released to a service provider.
market template	In PeopleSoft Enterprise Incentive Management, additional functionality that is specific to a given market or industry and is built on top of a product category.
match group	In PeopleSoft Receivables, a group of receivables items and matching offset items. The system creates match groups by using user-defined matching criteria for selected field values.
MCF server	Abbreviation for *PeopleSoft MultiChannel Framework server.* Comprises the universal queue server and the MCF log server. Both processes are started when *MCF Servers* is selected in an application server domain configuration.
merchandising activity	In PeopleSoft Promotions Management, a specific discount type that is associated with a trade promotion (such as off-invoice, billback or rebate, or lump-sum payment) that defines the performance that is required to receive the discount. In the industry, you may know this as an offer, a discount, a merchandising event, an event, or a tactic.
meta-SQL	Meta-SQL constructs expand into platform-specific Structured Query Language (SQL) substrings. They are used in functions that pass SQL strings, such as in SQL objects, the SQLExec function, and PeopleSoft Application Engine programs.
metastring	Metastrings are special expressions included in SQL string literals. The metastrings, prefixed with a percent (%) symbol, are included directly in the string literals. They expand at run time into an appropriate substring for the current database platform.
multibook	In PeopleSoft General Ledger, multiple ledgers having multiple-base currencies that are defined for a business unit, with the option to post a single transaction to all base currencies (all ledgers) or to only one of those base currencies (ledgers).
multicurrency	The ability to process transactions in a currency other than the business unit's base currency.
national allowance	In PeopleSoft Promotions Management, a promotion at the corporate level that is funded by nondiscretionary dollars. In the industry, you may know this as a national promotion, a corporate promotion, or a corporate discount.
node-oriented tree	A tree that is based on a detail structure, but the detail values are not used.
pagelet	Each block of content on the home page is called a pagelet. These pagelets display summary information within a small rectangular area on the page. The pagelet provide users with a snapshot of their most relevant PeopleSoft and non-PeopleSoft content.
participant	In PeopleSoft Enterprise Incentive Management, participants are recipients of the incentive compensation calculation process.
participant object	Each participant object may be related to one or more compensation objects. See also *compensation object.*
partner	A company that supplies products or services that are resold or purchased by the enterprise.
pay cycle	In PeopleSoft Payables, a set of rules that define the criteria by which it should select scheduled payments for payment creation.
pending item	In PeopleSoft Receivables, an individual receivable (such as an invoice, a credit memo, or a write-off) that has been entered in or created by the system, but hasn't been posted.

PeopleCode PeopleCode is a proprietary language, executed by the PeopleSoft application processor. PeopleCode generates results based upon existing data or user actions. By using business interlink objects, external services are available to all PeopleSoft applications wherever PeopleCode can be executed.

PeopleCode event An action that a user takes upon an object, usually a record field, that is referenced within a PeopleSoft page.

PeopleSoft Internet Architecture The fundamental architecture on which PeopleSoft 8 applications are constructed, consisting of a relational database management system (RDBMS), an application server, a web server, and a browser.

performance measurement In PeopleSoft Enterprise Incentive Management, a variable used to store data (similar to an aggregator, but without a predefined formula) within the scope of an incentive plan. Performance measures are associated with a plan calendar, territory, and participant. Performance measurements are used for quota calculation and reporting.

period context In PeopleSoft Enterprise Incentive Management, because a participant typically uses the same compensation plan for multiple periods, the period context associates a plan context with a specific calendar period and fiscal year. The period context references the associated plan context, thus forming a chain. Each plan context has a corresponding set of period contexts.

plan In PeopleSoft Sales Incentive Management, a collection of allocation rules, variables, steps, sections, and incentive rules that instruct the PeopleSoft Enterprise Incentive Management engine in how to process transactions.

plan context In PeopleSoft Enterprise Incentive Management, correlates a participant with the compensation plan and node to which the participant is assigned, enabling the PeopleSoft Enterprise Incentive Management system to find anything that is associated with the node and that is required to perform compensation processing. Each participant, node, and plan combination represents a unique plan context—if three participants are on a compensation structure, each has a different plan context. Configuration plans are identified by plan contexts and are associated with the participants that refer to them.

plan template In PeopleSoft Enterprise Incentive Management, the base from which a plan is created. A plan template contains common sections and variables that are inherited by all plans that are created from the template. A template may contain steps and sections that are not visible in the plan definition.

planned learning In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's planned learning activities and programs.

planning instance In PeopleSoft Supply Planning, a set of data (business units, items, supplies, and demands) constituting the inputs and outputs of a supply plan.

portal registry In PeopleSoft applications, the portal registry is a tree-like structure in which content references are organized, classified, and registered. It is a central repository that defines both the structure and content of a portal through a hierarchical, tree-like structure of folders useful for organizing and securing content references.

price list In PeopleSoft Enterprise Pricer, enables you to select products and conditions for which the price list applies to a transaction. During a transaction, the system either determines the product price based on the predefined search hierarchy for the transaction or uses the product's lowest price on any associated, active price lists. This price is used as the basis for any further discounts and surcharges.

price rule In PeopleSoft Enterprise Pricer, defines the conditions that must be met for adjustments to be applied to the base price. Multiple rules can apply when conditions of each rule are met.

price rule condition In PeopleSoft Enterprise Pricer, selects the price-by fields, the values for the price-by fields, and the operator that determines how the price-by fields are related to the transaction.

price rule key In PeopleSoft Enterprise Pricer, defines the fields that are available to define price rule conditions (which are used to match a transaction) on the price rule.

process category In PeopleSoft Process Scheduler, processes that are grouped for server load balancing and prioritization.

process group In PeopleSoft Financials, a group of application processes (performed in a defined order) that users can initiate in real time, directly from a transaction entry page.

process definition Process definitions define each run request.

process instance A unique number that identifies each process request. This value is automatically incremented and assigned to each requested process when the process is submitted to run.

process job You can link process definitions into a job request and process each request serially or in parallel. You can also initiate subsequent processes based on the return code from each prior request.

process request A single run request, such as a Structured Query Report (SQR), a COBOL or Application Engine program, or a Crystal report that you run through PeopleSoft Process Scheduler.

process run control A PeopleTools variable used to retain PeopleSoft Process Scheduler values needed at runtime for all requests that reference a run control ID. Do not confuse these with application run controls, which may be defined with the same run control ID, but only contain information specific to a given application process request.

product category In PeopleSoft Enterprise Incentive Management, indicates an application in the Enterprise Incentive Management suite of products. Each transaction in the PeopleSoft Enterprise Incentive Management system is associated with a product category.

programs In PeopleSoft Enterprise Learning Management, a high-level grouping that guides the learner along a specific learning path through sections of catalog items. PeopleSoft Enterprise Learning Systems provides two types of programs—curricula and certifications.

progress log In PeopleSoft Services Procurement, tracks deliverable-based projects. This is similar to the time sheet in function and process. The service provider contact uses the progress log to record and submit progress on deliverables. The progress can be logged by the activity that is performed, by the percentage of work that is completed, or by the completion of milestone activities that are defined for the project.

project transaction In PeopleSoft Project Costing, an individual transaction line that represents a cost, time, budget, or other transaction row.

promotion In PeopleSoft Promotions Management, a trade promotion, which is typically funded from trade dollars and used by consumer products manufacturers to increase sales volume.

publishing In PeopleSoft Enterprise Incentive Management, a stage in processing that makes incentive-related results available to participants.

record group A set of logically and functionally related control tables and views. Record groups help enable TableSet sharing, which eliminates redundant data entry. Record groups ensure that TableSet sharing is applied consistently across all related tables and views.

record input VAT flag Abbreviation for *record input value-added tax flag*. Within PeopleSoft Purchasing, Payables, and General Ledger, this flag indicates that you are recording input VAT

	on the transaction. This flag, in conjunction with the record output VAT flag, is used to determine the accounting entries created for a transaction and to determine how a transaction is reported on the VAT return. For all cases within Purchasing and Payables where VAT information is tracked on a transaction, this flag is set to Yes. This flag is not used in PeopleSoft Order Management, Billing, or Receivables, where it is assumed that you are always recording only output VAT, or in PeopleSoft Expenses, where it is assumed that you are always recording only input VAT.
record output VAT flag	Abbreviation for *record output value-added tax flag*. See *record input VAT flag*.
reference data	In PeopleSoft Sales Incentive Management, system objects that represent the sales organization, such as territories, participants, products, customers, channels, and so on.
reference object	In PeopleSoft Enterprise Incentive Management, this dimension-type object further defines the business. Reference objects can have their own hierarchy (for example, product tree, customer tree, industry tree, and geography tree).
reference transaction	In commitment control, a reference transaction is a source transaction that is referenced by a higher-level (and usually later) source transaction, in order to automatically reverse all or part of the referenced transaction's budget-checked amount. This avoids duplicate postings during the sequential entry of the transaction at different commitment levels. For example, the amount of an encumbrance transaction (such as a purchase order) will, when checked and recorded against a budget, cause the system to concurrently reference and relieve all or part of the amount of a corresponding pre-encumbrance transaction, such as a purchase requisition.
regional sourcing	In PeopleSoft Purchasing, provides the infrastructure to maintain, display, and select an appropriate vendor and vendor pricing structure that is based on a regional sourcing model where the multiple ship to locations are grouped. Sourcing may occur at a level higher than the ship to location.
relationship object	In PeopleSoft Enterprise Incentive Management, these objects further define a compensation structure to resolve transactions by establishing associations between compensation objects and business objects.
remote data source data	Data that is extracted from a separate database and migrated into the local database.
REN server	Abbreviation for *real-time event notification server* in PeopleSoft MultiChannel Framework.
requester	In PeopleSoft eSettlements, an individual who requests goods or services and whose ID appears on the various procurement pages that reference purchase orders.
role	Describes how people fit into PeopleSoft Workflow. A role is a class of users who perform the same type of work, such as clerks or managers. Your business rules typically specify what user role needs to do an activity.
role user	A PeopleSoft Workflow user. A person's role user ID serves much the same purpose as a user ID does in other parts of the system. PeopleSoft Workflow uses role user IDs to determine how to route worklist items to users (through an email address, for example) and to track the roles that users play in the workflow. Role users do not need PeopleSoft user IDs.
roll up	In a tree, to roll up is to total sums based on the information hierarchy.
run control	A run control is a type of online page that is used to begin a process, such as the batch processing of a payroll run. Run control pages generally start a program that manipulates data.
run control ID	A unique ID to associate each user with his or her own run control table entries.

run-level context In PeopleSoft Enterprise Incentive Management, associates a particular run (and batch ID) with a period context and plan context. Every plan context that participates in a run has a separate run-level context. Because a run cannot span periods, only one run-level context is associated with each plan context.

search query You use this set of objects to pass a query string and operators to the search engine. The search index returns a set of matching results with keys to the source documents.

section In PeopleSoft Enterprise Incentive Management, a collection of incentive rules that operate on transactions of a specific type. Sections enable plans to be segmented to process logical events in different sections.

security event In commitment control, security events trigger security authorization checking, such as budget entries, transfers, and adjustments; exception overrides and notifications; and inquiries.

serial genealogy In PeopleSoft Manufacturing, the ability to track the composition of a specific, serial-controlled item.

serial in production In PeopleSoft Manufacturing, enables the tracing of serial information for manufactured items. This is maintained in the Item Master record.

session In PeopleSoft Enterprise Learning Management, a single meeting day of an activity (that is, the period of time between start and finish times within a day). The session stores the specific date, location, meeting time, and instructor. Sessions are used for scheduled training.

session template In PeopleSoft Enterprise Learning Management, enables you to set up common activity characteristics that may be reused while scheduling a PeopleSoft Enterprise Learning Management activity—characteristics such as days of the week, start and end times, facility and room assignments, instructors, and equipment. A session pattern template can be attached to an activity that is being scheduled. Attaching a template to an activity causes all of the default template information to populate the activity session pattern.

setup relationship In PeopleSoft Enterprise Incentive Management, a relationship object type that associates a configuration plan with any structure node.

share driver expression In PeopleSoft Business Planning, a named planning method similar to a driver expression, but which you can set up globally for shared use within a single planning application or to be shared between multiple planning applications through PeopleSoft Enterprise Warehouse.

single signon With single signon, users can, after being authenticated by a PeopleSoft application server, access a second PeopleSoft application server without entering a user ID or password.

source transaction In commitment control, any transaction generated in a PeopleSoft or third-party application that is integrated with commitment control and which can be checked against commitment control budgets. For example, a pre-encumbrance, encumbrance, expenditure, recognized revenue, or collected revenue transaction.

SpeedChart A user-defined shorthand key that designates several ChartKeys to be used for voucher entry. Percentages can optionally be related to each ChartKey in a SpeedChart definition.

SpeedType A code representing a combination of ChartField values. SpeedTypes simplify the entry of ChartFields commonly used together.

staging A method of consolidating selected partner offerings with the offerings from the enterprise’s other partners.

statutory account Account required by a regulatory authority for recording and reporting financial results. In PeopleSoft, this is equivalent to the Alternate Account (ALTACCT) ChartField.

step In PeopleSoft Sales Incentive Management, a collection of sections in a plan. Each step corresponds to a step in the job run.

storage level In PeopleSoft Inventory, identifies the level of a material storage location. Material storage locations are made up of a business unit, a storage area, and a storage level. You can set up to four storage levels.

subcustomer qualifier A value that groups customers into a division for which you can generate detailed history, aging, events, and profiles.

Summary ChartField You use summary ChartFields to create summary ledgers that roll up detail amounts based on specific detail values or on selected tree nodes. When detail values are summarized using tree nodes, summary ChartFields must be used in the summary ledger data record to accommodate the maximum length of a node name (20 characters).

summary ledger An accounting feature used primarily in allocations, inquiries, and PS/nVision reporting to store combined account balances from detail ledgers. Summary ledgers increase speed and efficiency of reporting by eliminating the need to summarize detail ledger balances each time a report is requested. Instead, detail balances are summarized in a background process according to user-specified criteria and stored on summary ledgers. The summary ledgers are then accessed directly for reporting.

summary time period In PeopleSoft Business Planning, any time period (other than a base time period) that is an aggregate of other time periods, including other summary time periods and base time periods, such as quarter and year total.

summary tree A tree used to roll up accounts for each type of report in summary ledgers. Summary trees enable you to define trees on trees. In a summary tree, the detail values are really nodes on a detail tree or another summary tree (known as the *basis* tree). A summary tree structure specifies the details on which the summary trees are to be built.

syndicate To distribute a production version of the enterprise catalog to partners.

system function In PeopleSoft Receivables, an activity that defines how the system generates accounting entries for the general ledger.

TableSet A means of sharing similar sets of values in control tables, where the actual data values are different but the structure of the tables is the same.

TableSet sharing Shared data that is stored in many tables that are based on the same TableSets. Tables that use TableSet sharing contain the SETID field as an additional key or unique identifier.

target currency The value of the entry currency or currencies converted to a single currency for budget viewing and inquiry purposes.

template A template is HTML code associated with a web page. It defines the layout of the page and also where to get HTML for each part of the page. In PeopleSoft, you use templates to build a page by combining HTML from a number of sources. For a PeopleSoft portal, all templates must be registered in the portal registry, and each content reference must be assigned a template.

territory In PeopleSoft Sales Incentive Management, hierarchical relationships of business objects, including regions, products, customers, industries, and participants.

TimeSpan A relative period, such as year-to-date or current period, that can be used in various PeopleSoft General Ledger functions and reports when a rolling time frame, rather

	than a specific date, is required. TimeSpans can also be used with flexible formulas in PeopleSoft Projects.
trace usage	In PeopleSoft Manufacturing, enables the control of which components will be traced during the manufacturing process. Serial- and lot-controlled components can be traced. This is maintained in the Item Master record.
transaction allocation	In PeopleSoft Enterprise Incentive Management, the process of identifying the owner of a transaction. When a raw transaction from a batch is allocated to a plan context, the transaction is duplicated in the PeopleSoft Enterprise Incentive Management transaction tables.
transaction state	In PeopleSoft Enterprise Incentive Management, a value assigned by an incentive rule to a transaction. Transaction states enable sections to process only transactions that are at a specific stage in system processing. After being successfully processed, transactions may be promoted to the next transaction state and "picked up" by a different section for further processing.
Translate table	A system edit table that stores codes and translate values for the miscellaneous fields in the database that do not warrant individual edit tables of their own.
tree	The graphical hierarchy in PeopleSoft systems that displays the relationship between all accounting units (for example, corporate divisions, projects, reporting groups, account numbers) and determines roll-up hierarchies.
unclaimed transaction	In PeopleSoft Enterprise Incentive Management, a transaction that is not claimed by a node or participant after the allocation process has completed, usually due to missing or incomplete data. Unclaimed transactions may be manually assigned to the appropriate node or participant by a compensation administrator.
universal navigation header	Every PeopleSoft portal includes the universal navigation header, intended to appear at the top of every page as long as the user is signed on to the portal. In addition to providing access to the standard navigation buttons (like Home, Favorites, and signoff) the universal navigation header can also display a welcome message for each user.
user interaction object	In PeopleSoft Sales Incentive Management, used to define the reporting components and reports that a participant can access in his or her context. All Sales Incentive Management user interface objects and reports are registered as user interaction objects. User interaction objects can be linked to a compensation structure node through a compensation relationship object (individually or as groups).
variable	In PeopleSoft Sales Incentive Management, the intermediate results of calculations. Variables hold the calculation results and are then inputs to other calculations. Variables can be plan variables that persist beyond the run of an engine or local variables that exist only during the processing of a section.
VAT exception	Abbreviation for *value-added tax exception.* A temporary or permanent exemption from paying VAT that is granted to an organization. This terms refers to both VAT exoneration and VAT suspension.
VAT exempt	Abbreviation for *value-added tax exempt.* Describes goods and services that are not subject to VAT. Organizations that supply exempt goods or services are unable to recover the related input VAT. This is also referred to as exempt without recovery.
VAT exoneration	Abbreviation for *value-added tax exoneration.* An organization that has been granted a permanent exemption from paying VAT due to the nature of that organization.
VAT suspension	Abbreviation for *value-added tax suspension.* An organization that has been granted a temporary exemption from paying VAT.
warehouse	A PeopleSoft data warehouse that consists of predefined ETL maps, data warehouse tools, and DataMart definitions.

work order In PeopleSoft Services Procurement, enables an enterprise to create resource-based and deliverable-based transactions that specify the basic terms and conditions for hiring a specific service provider. When a service provider is hired, the service provider logs time or progress against the work order.

worksheet A way of presenting data through a PeopleSoft Business Analysis Modeler interface that enables users to do in-depth analysis using pivoting tables, charts, notes, and history information.

worklist The automated to-do list that PeopleSoft Workflow creates. From the worklist, you can directly access the pages you need to perform the next action, and then return to the worklist for another item.

XML schema An XML definition that standardizes the representation of application messages, component interfaces, or business interlinks.

yield by operation In PeopleSoft Manufacturing, the ability to plan the loss of a manufactured item on an operation-by-operation basis.

zero-rated VAT Abbreviation for *zero-rated value-added tax.* A VAT transaction with a VAT code that has a tax percent of zero. Used to track taxable VAT activity where no actual VAT amount is charged. Organizations that supply zero-rated goods and services can still recover the related input VAT. This is also referred to as exempt with recovery.

Index

A

T