PeopleSoft®

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Optimization Framework

June 2004

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Optimization Framework
SKU PT845OPT-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 Optimization Framework Preface

This PeopleBook describes PeopleSoft Optimization Framework.

PeopleSoft Optimization Framework

PeopleSoft Optimization Framework provides a foundation for building applications that use the optimization-based, decision-making capability in the PeopleTools environment. This PeopleBook is written for PeopleSoft application developers who write PeopleCode to use optimization transactions.

CHAPTER 1

Getting Started with PeopleSoft Optimization Framework

PeopleSoft Optimization Framework provides a foundation for building applications that use the optimization-based, decision-making capability in the PeopleTools environment. This chapter provides an overview of the PeopleSoft Optimization Framework and discusses how to implement the optimization engines.

PeopleSoft Optimization Framework Overview

This section provides an overview of the conceptual information available about the PeopleSoft Optimization Framework.

- Understanding PeopleSoft Optimization discusses optimization, the framework components and architecture, as well as doing optimization -based development.
- Designing Problem Type Definitions provides overviews of problem type definitions and optimization application records. It also discusses how to use these items and develop your own application-based optimization.
- Optimization PeopleCode contains the reference material for the PeopleCode used in PeopleSoft Optimization Framework, as well as considerations for creating optimization PeopleCode programs.
- Administering Optimization Server Components provides an overview of optimization administration and discusses configuring the optimization engines.

See Also

Chapter 2, "Understanding PeopleSoft Optimization Framework," page 3

Chapter 3, "Designing Problem Type Definitions," Creating Problem Type Definitions, page 10

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Optimization PeopleCode"

Chapter 5, "Administering Optimization Server Components," page 101

PeopleSoft Optimization Framework Implementation

The functionality to use the PeopleSoft Optimization Framework, as well as to create your own Optimization plug-in (OPI) is delivered as part of standard PeopleSoft PeopleTools that are provided with all PeopleSoft products.

Several activities must be completed before you can use the PeopleSoft Optimization Framework in your implementation.

- Install your PeopleSoft application according to the installation guide for your database type. See *The PeopleSoft Installation Guide* for your platform and product line.
- Establish a user profile that gives you access to PeopleSoft Application Designer and any other processes you will use. See *PeopleTools 8.45 Security Administration*.
- Follow the general overview and instructions in this document to design your application to take advantage of the PeopleSoft Optimization Framework, populate the appropriate records, build the application pages, retrieve the result data, as well as configure the application server and the optimization engines.

See Also

Chapter 2, “Understanding PeopleSoft Optimization Framework,” page 3

Chapter 5, “Administering Optimization Server Components,” page 101

CHAPTER 2

Understanding PeopleSoft Optimization Framework

This chapter discusses:

- Optimization.
- PeopleSoft Optimization Framework components.
- PeopleSoft Optimization Framework system architecture.
- Optimization-based application development.

Optimization

In the context of PeopleSoft Optimization Framework, *optimization* means deciding on the best course of action given a range of alternatives. You use PeopleSoft Optimization Framework and the PeopleTools environment to build applications that use optimization-based decision-making. PeopleSoft Optimization Framework enables applications to specify their business objectives, define the conditions, and set resource constraints. PeopleSoft Optimization Framework then applies advanced mathematical modeling and solution techniques to find solutions that fit input criteria. In contrast to sequential, query-based applications, which require users to analyze criteria and make decisions one by one, the solution generated by optimization exceeds, or at least matches, a solution generated by a person.

PeopleSoft Optimization Framework Components

PeopleSoft Optimization Framework contains the following main elements:

- Optimization application tables.

  PeopleSoft database tables that store source data, result data, control parameters, and user state information.

- Optimization engine.

  An instance of the optimization engine is a process managed by a PeopleSoft application server. The optimization engine has a generic interface to bind with different optimization plug-ins to provide a variety of optimization services. It also brings data from the optimization application tables into memory. This in-memory data is synchronized with the database changes with each optimization transaction.

- Optimization dispatcher.

  Within the application server, the optimization dispatcher provides a generic interface for application programmers to use PeopleCode to access the optimization engine.

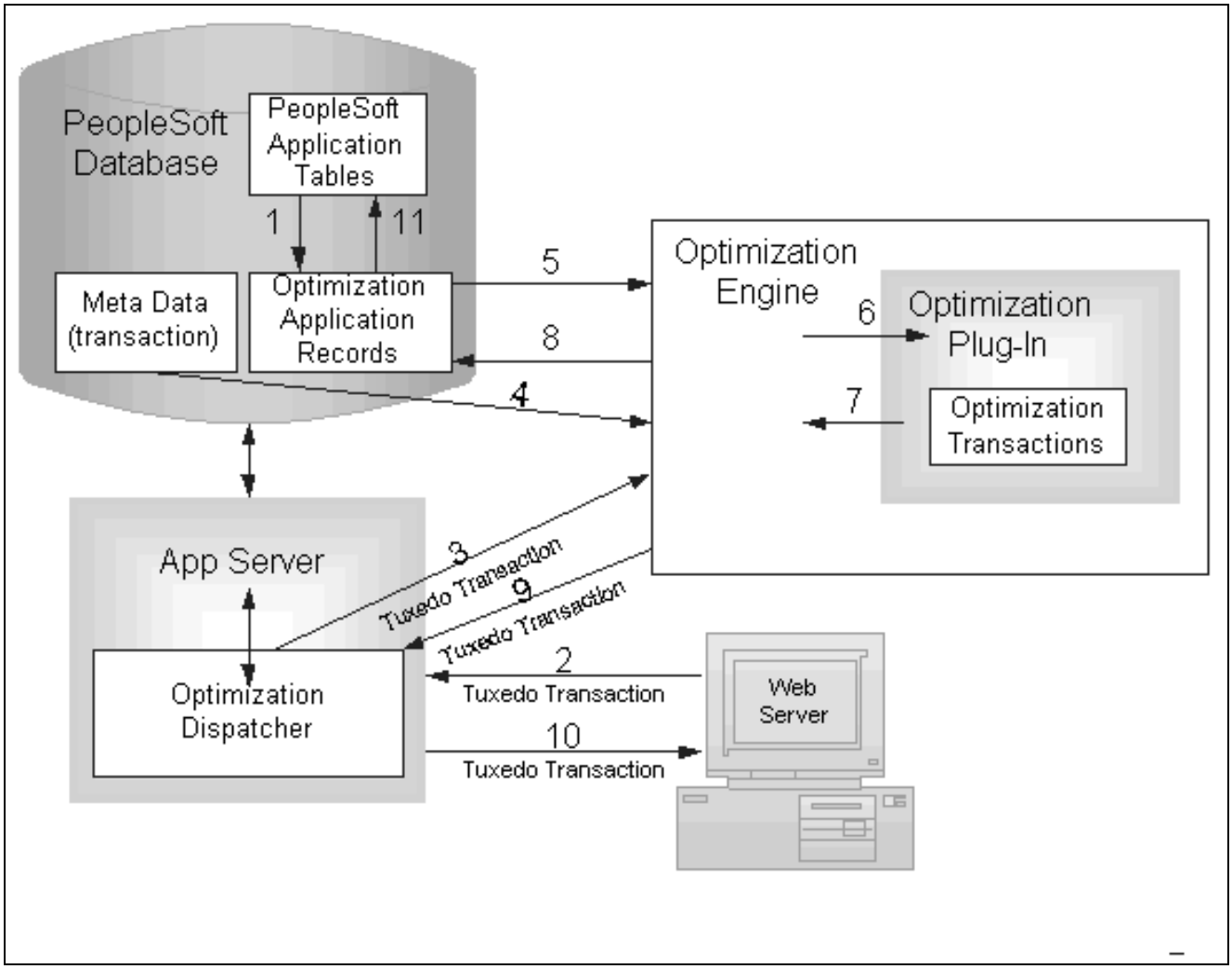
- Optimization plug-in (OPI).

  An OPI is created specifically for optimization-based applications, such as consultant scheduling or supply chain planning and scheduling. The application knowledge and business logic of an optimization problem resides in the OPI. The OPI implements the optimization transactions that solve the problem using the source data as input and generating the result data as output. If your application is delivered with the *Optimization PeopleCode* plug-in, you'll be able to adapt the plug-in to a variety of different optimization tasks.

Note. An OPI is created by PeopleTools development with support from PeopleSoft application development. An OPI is provided with the installed PeopleSoft applications that use PeopleSoft Optimization Framework. There is no OPI in the PeopleTools installation. Your PeopleSoft application documentation discusses the available plugins and their required implementation steps and parameters.

PeopleSoft Optimization Framework System Architecture

The following diagram illustrates PeopleSoft Optimization Framework architecture components and shows the sequence of use during a typical optimization transaction:



PeopleSoft Optimization Framework architecture

When an optimization-based application runs, the following actions occur:

1. The source data is loaded from PeopleSoft application tables into the optimization application records.

   Depending on the amount of data, this can typically be done as a batch job.

2. A web server sends a request through BEA Tuxedo to have the application server perform a PeopleSoft transaction using optimization.

3. Upon receiving the request, the optimization dispatcher, within the application server, locates the correct optimization engine and sends the optimization transaction to it through BEA Tuxedo.

4. The optimization engine gets the metadata (optimization transaction name, parameters, and data types of the parameters) from the problem type definition in the PeopleSoft application database.

   It uses this information to check the integrity of the optimization transaction request. It also synchronizes the data in memory with changes in the optimization application tables.

5. The optimization engine reads the changed data (all the data, if this is the first time the data is being read) from the optimization application records into memory.

6. The optimization engine loads the appropriate OPI and passes the optimization transaction request to it.

   The OPI is loaded during the first request to the optimization engine. It remains loaded until the optimization engine is shut down.

7. The OPI processes the transaction and provides result data in the form of output parameters to the optimization engine.

   The OPI might also change data in memory to be saved to the database.

8. The optimization engine writes the changed data in memory to the optimization application tables.

9. The optimization engine returns the result data to the optimization dispatcher.

10. The application server completes the PeopleSoft transaction with the result data and returns a success code to the user and to the web server through BEA Tuxedo.

11. Once the user is satisfied with the optimization result data, the result data can be copied from the optimization application tables to the PeopleSoft application tables.

Optimization-Based Application Development

To build an optimization-based application:

1. Design the problem type definition.

   Define the structure of the optimization application records and the specifications for the optimization transactions that you need for your application. Using PeopleSoft Application Designer:

   a. Create record definitions for the optimization application records and build them to create the database tables.

   b. Create a problem type definition, including the record definitions that you created, and the specifications for the optimization transactions.

   c. You might also need to insert one or more *optimization models* into the problem type definition. Optimization models are developed specifically for, and delivered with, your PeopleSoft application. Each optimization model is a mathematical representation of the business problem for the optimization engine to solve.

2. Populate the application records with appropriate source data.

Using standard tools (such as PeopleCode, PeopleSoft Application Engine, and PeopleSoft Integration Broker), provide a mechanism to populate the optimization application records with source data. You can also use PeopleSoft application records directly instead of creating special optimization application records. By accessing the tables directly, you use fewer computer resources. However, accessing the application tables directly increases the dependency between the application design and the OPI design.

3. Build the application pages.

   Using PeopleSoft Application Designer, build pages using the optimization application records to enable users to edit or view the source and result data and to interact with the optimization application. These pages use the PeopleCode OptEngine class, provided by the optimization dispatcher, to send optimization transactions to the optimization engine. Building pages for optimization applications uses the same process as building pages for any PeopleSoft application.

4. Retrieve the result data.

   Using standard PeopleTools, provide a mechanism to retrieve the result data in the optimization application records and copy it to the PeopleSoft application tables.

Note. If you rename any records or record fields that are used by your optimization-based application, the problem type and optimization model definitions that use the record or field automatically reflect your changes. However, you must also ensure that any PeopleCode program, Application Engine program, or other tools account for those changes as well.

CHAPTER 3

Designing Problem Type Definitions

This chapter provides overviews of problem type definitions and optimization application record design and discusses how to:

- Create and build optimization records.
- Create problem type definitions.
- Run the optimization system audit.
- Change existing problem type definitions.
- Administer optimization engines.
- Administer optimization tables.
- Update solver licenses.

Understanding Problem Type Definitions

A problem type definition groups the optimization application records, the optimization transactions, and the OPI together as one entity. The optimization application records contain the data stored in the database. The data is populated into memory in the optimization engine. The optimization transactions define the interface between the application server and the OPI, which performs the optimization computation. Use PeopleSoft Application Designer to create the problem type definition for an optimization application.

An Optimization Problem Example

To illustrate the steps of creating an optimization-based application, consider the following example: Create an optimal exercise schedule that makes use of exercise machine availability and satisfies individuals' exercise preferences. To create an optimization application for this problem, you need input data about:

- Exercises that burn a set number of calories per minute.
- People who know how long they want to exercise and how many calories they want to burn.

The goal of your application is to generate a list containing an exercise and the duration of exercise appropriate to each person, based on the input data.

To implement the problem type definition for this example, you would:

1. Create and populate a set of records containing the input data about the exercises and the participants.

   These are the optimization application records for this application.

2. Define a set of optimization transactions and their parameters that, when implemented, process the optimization application records to achieve the goal.

Note. For this example, assume that an OPI (QEOPT.DLL) already exists that implements these transactions.

Understanding Optimization Application Record Design

This section discusses:

- Optimization application records.
- Scenario management.

Optimization Application Records

You use PeopleSoft Application Designer to design optimization application records to contain source data, result data, and other data. You also decide how the optimization engine uses these records for synchronization. For each record that you create, decide:

- Which data fields the record should contain.

  Among other data, these records contain the data from the PeopleSoft application database that is used in the optimization process.

- How the optimization engine uses the record for synchronization.

  If the record is read once, the optimization engine reads this data during the initial load only. If the record is readable, the optimization engine checks for updates with every optimization transaction. If the record is writable, the optimization engine is allowed to modify the data in the database. All records except read-once records must have a VERSION field.

- Whether the record should be scenario-managed.

  A record should be scenario-managed if it contains data pertaining to multiple problem instances. Such records must have a PROBINST key field, which the optimization engine uses as an additional key for storing and retrieving multiple solutions.

Scenario Management

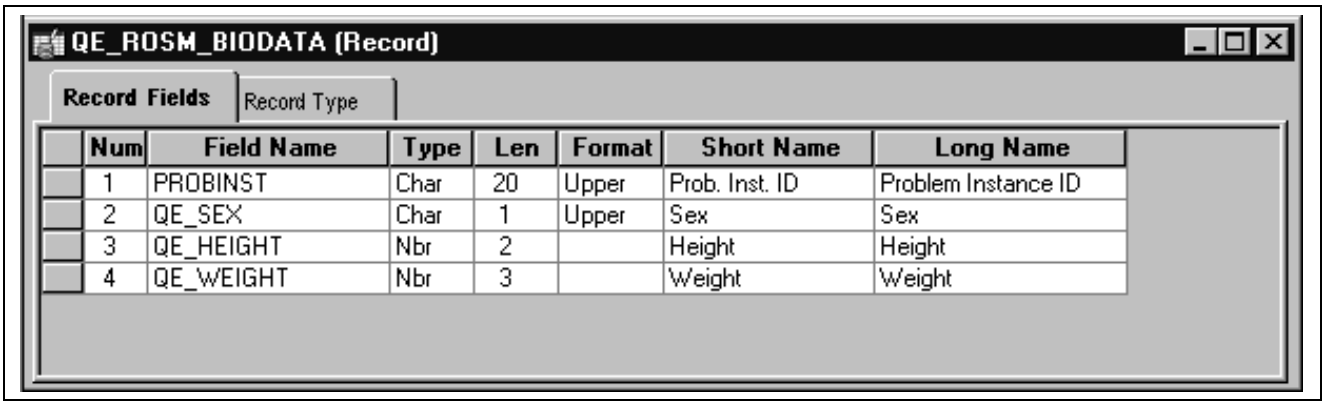
In PeopleSoft Optimization Framework, scenario management is the mechanism to manage different source and result data sets using the same tables. A set of source data and associated result data is called a problem instance. You can break down large optimization problems into smaller, more manageable problems (or problem instances) that can each be solved independently. Individual problem instances can share common data.

This concept can be extended to what-if scenarios to plan for potential business situations. Separate problem instances can be created with what-if data and solved using optimization separately, without fear of affecting live data.

In terms of the exercise example, any number of people might want exercise schedules using the optimization application. Exercise goal data and the optimization-generated exercise schedule data are unique to each person. However, different people share the same set of exercise machines. In this case, it is logical to treat the generation of an individual person's exercise schedule as a separate problem instance.

In the exercise example, you would mark the data that is specific to each person (such as exercise goals and exercise schedules) as scenario-managed, and the data that is shared by all people (such as exercise machines) as non-scenario-managed. All scenario-managed records must include the PROBINST field as part of the primary key. This 20-character field identifies data specific to a problem instance. During runtime, the optimization engine loads data for scenario-managed records based on the user-specified value for the PROBINST field. At any moment, the optimization engine contains data for only one problem instance.

The following record, QE_ROSM_BIODATA, contains the name of a person who exercises, and physical data about the person. This record is read once and scenario-managed. Notice the use of the PROBINST field.



Num	Field Name	Type	Len	Format	Short Name	Long Name
1	PROBINST	Char	20	Upper	Prob. Inst. ID	Problem Instance ID
2	QE_SEX	Char	1	Upper	Sex	Sex
3	QE_HEIGHT	Nbr	2		Height	Height
4	QE_WEIGHT	Nbr	3		Weight	Weight

QE_ROSM_BIODATA record

Creating and Building Optimization Records

To create and build optimization application records:

1. Create the optimization application record definitions using PeopleSoft Application Designer.
2. For every optimization application record that is readable, create an optimization delete record by cloning the optimization application record.

   Clone the record by performing a Save As operation on the optimization application record and renaming the optimization delete record to be similar to the original optimization application record. Use a naming convention for all optimization delete records. For example, the optimization delete record for the record QE_R_HOLIDAYS might be named QE_R_HOLIDAYDEL.

   Alternately, use a sub-record definition that's shared by the optimization application record and the delete record.
3. For every optimization application record that is readable, associate that record with its optimization delete record.
   a. Open the optimization application record.
   b. Select File, Definition Properties.
   c. Select the Use tab in the Record Properties dialog box.
   d. Enter the name of the optimization delete record in the Optimization Delete Record field.
4. Open (or create) a project and insert all the optimization application records and optimization delete records into the project.
5. Create the tables from these records.
   a. Select Build, Project.

      The Build dialog box appears, showing the optimization application records and optimization delete records in the project.
   b. Select the Create Tables check box, and make sure that the Create Triggers check box is clear.
   c. Click the Build button.

6. Create optimization database triggers from these records.
    a. Select Build, Project.

       The Build page appears, showing the optimization application records and optimization delete records in the project.

    b. Select the Create Triggers check box.

    c. Click the Build button.

Note. Optimization delete records may be used by several problem types. When a record is deleted from a problem type, the associated delete record is not needed if this record is not used elsewhere.

Creating Problem Type Definitions

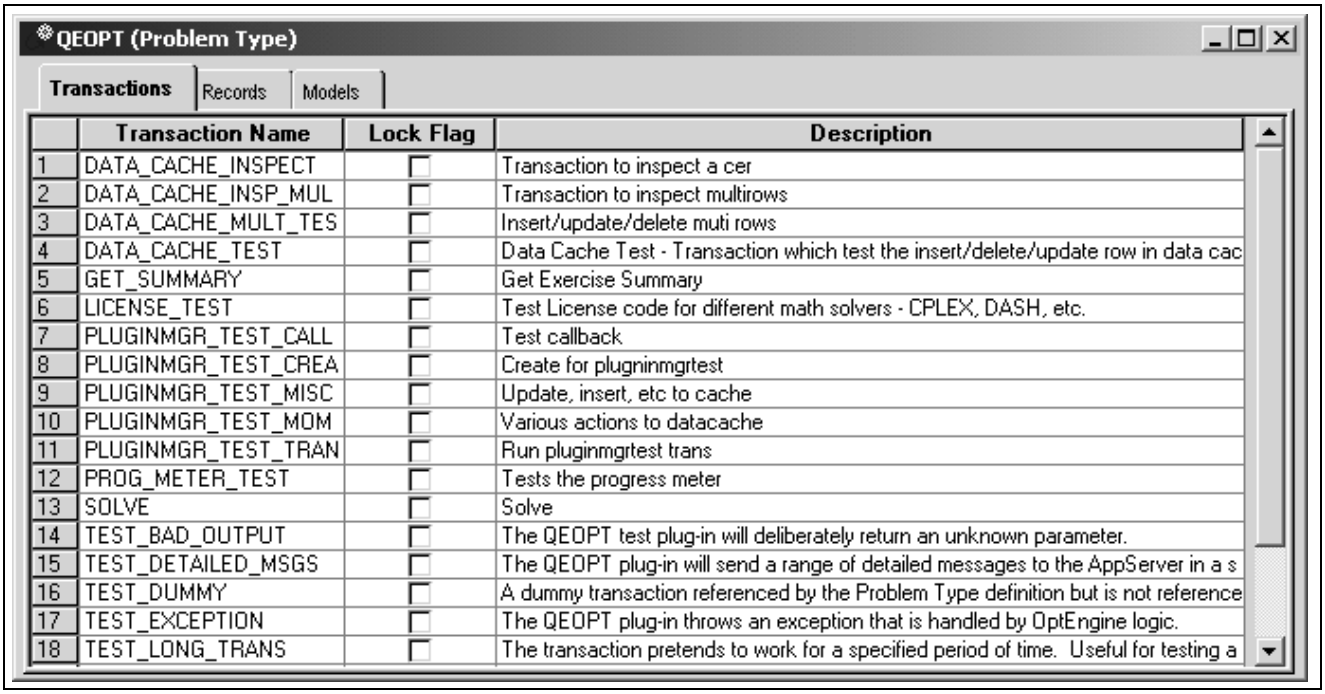
This section discusses how to:

- Define a problem type.
- Configure problem type records.
- Configure problem type models.
- Configure problem type transactions.

Note. When working with problem type definitions, you can use the typical drag-and-drop features offered by PeopleSoft Application Designer. For example, you can drag record definitions and drop them into the problem type record list, which is maintained on the Record tab of the problem type definition.

Defining a Problem Type

In PeopleSoft Application Designer, select File, New, Problem Type. A new problem type definition appears, containing tabs for transactions, records, and models. The definition combines these items with an OPI to form the basis of an optimization application.

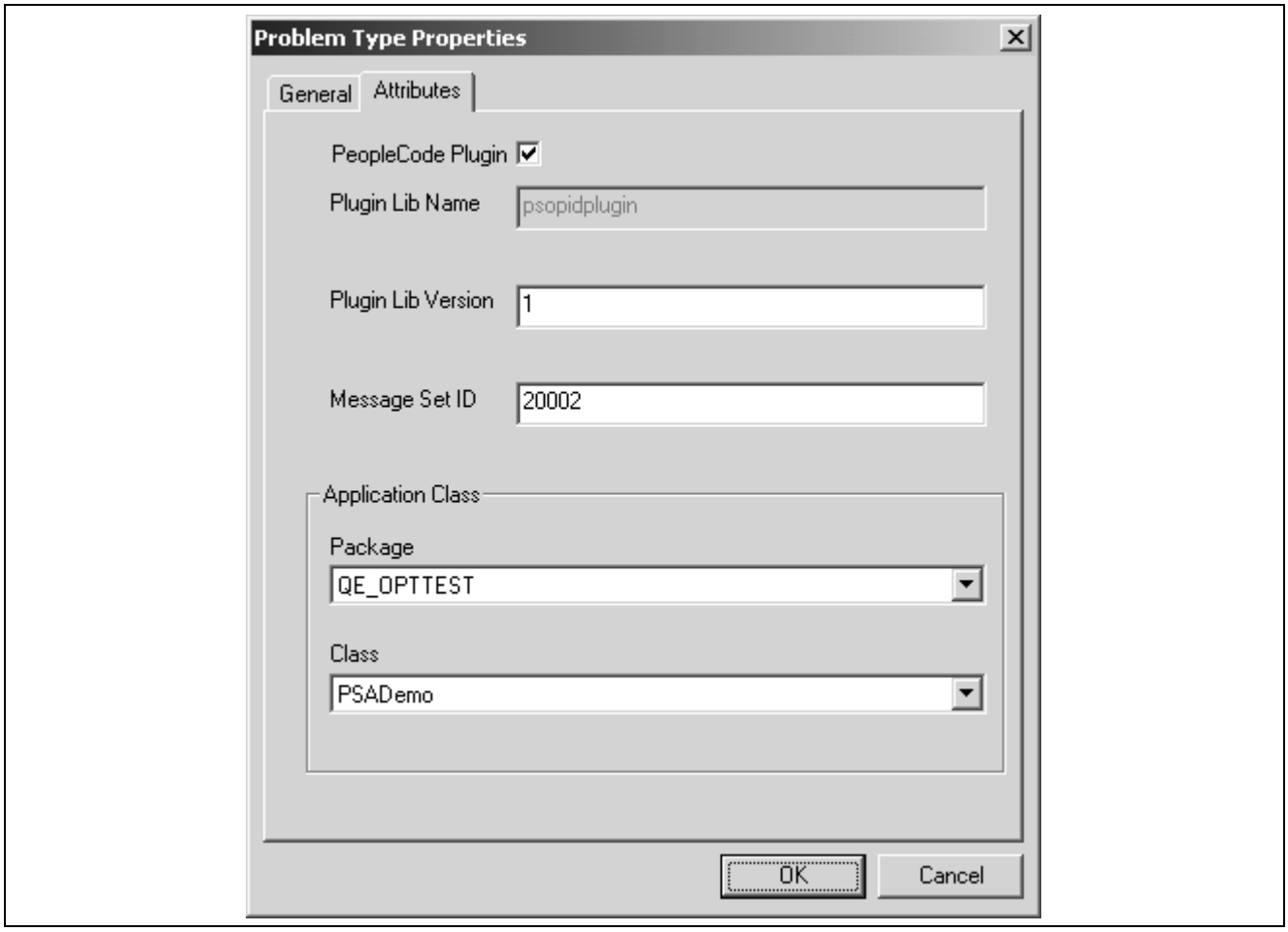


	Transaction Name	Lock Flag	Description
1	DATA_CACHE_INSPECT	☐	Transaction to inspect a cer
2	DATA_CACHE_INSP_MUL	☐	Transaction to inspect multirows
3	DATA_CACHE_MULT_TES	☐	Insert/update/delete muti rows
4	DATA_CACHE_TEST	☐	Data Cache Test - Transaction which test the insert/delete/update row in data cac
5	GET_SUMMARY	☐	Get Exercise Summary
6	LICENSE_TEST	☐	Test License code for different math solvers - CPLEX, DASH, etc.
7	PLUGINMGR_TEST_CALL	☐	Test callback
8	PLUGINMGR_TEST_CREA	☐	Create for plugninmgrtest
9	PLUGINMGR_TEST_MISC	☐	Update, insert, etc to cache
10	PLUGINMGR_TEST_MOM	☐	Various actions to datacache
11	PLUGINMGR_TEST_TRAN	☐	Run pluginmgrtest trans
12	PROG_METER_TEST	☐	Tests the progress meter
13	SOLVE	☐	Solve
14	TEST_BAD_OUTPUT	☐	The QEOPT test plug-in will deliberately return an unknown parameter.
15	TEST_DETAILED_MSGS	☐	The QEOPT plug-in will send a range of detailed messages to the AppServer in a s
16	TEST_DUMMY	☐	A dummy transaction referenced by the Problem Type definition but is not reference
17	TEST_EXCEPTION	☐	The QEOPT plug-in throws an exception that is handled by OptEngine logic.
18	TEST_LONG_TRANS	☐	The transaction pretends to work for a specified period of time. Useful for testing a

Problem Type - transactions tab

To complete the problem type definition, you'll find it most useful to configure the problem type properties, then insert and configure the records, the optimization models, and the transactions, in that order.

Access the Problem Type Properties dialog box, and select the Attributes tab.



Problem Type Properties - Attributes tab

PeopleCode Plugin

Select to indicate that the problem type should use the Optimization PeopleCode plug-in. *Psopidplugin* is automatically entered in the Plugin Lib Name field, which is read-only.

If you use this plug-in, you must also use the Package and Class fields to specify an application class that's been developed to adapt the Optimization PeopleCode plug-in to your optimization application.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," Invoking the Optimization PeopleCode Plug-In.

Plugin Lib Name (plug-in library name)

Enter the name of the OPI library. Enter only the portion of the name that is specific to this library. Ignore operating system-specific prefixes (such as lib) and suffixes (such as .dll). In the exercise example, on Microsoft Windows, the library is libqeopt.dll. You would enter only *qeopt* here.

If you selected the PeopleCode Plugin check box, this field contains the value *psopidplugin*, and is read-only.

Plugin Lib Version(plug-in library version)

Enter the application release version of the plug-in. The optimization engine uses this to confirm that the correct version of the plug-in library is used at runtime.

Message Set ID Enter the message set ID in the message catalog containing the messages for the optimization application. The OPI uses this to access messages from the message catalog.

Package If you selected the PeopleCode Plugin check box, you must specify here the application package containing the application class to use with the Optimization PeopleCode plug-in for your optimization application.

Class If you selected the PeopleCode Plugin check box, you must specify here the application class containing the optimization PeopleCode program to use with the Optimization PeopleCode plug-in for your optimization application. This class must be a subclass of the PT_OPT_BASE:OptBase application class.

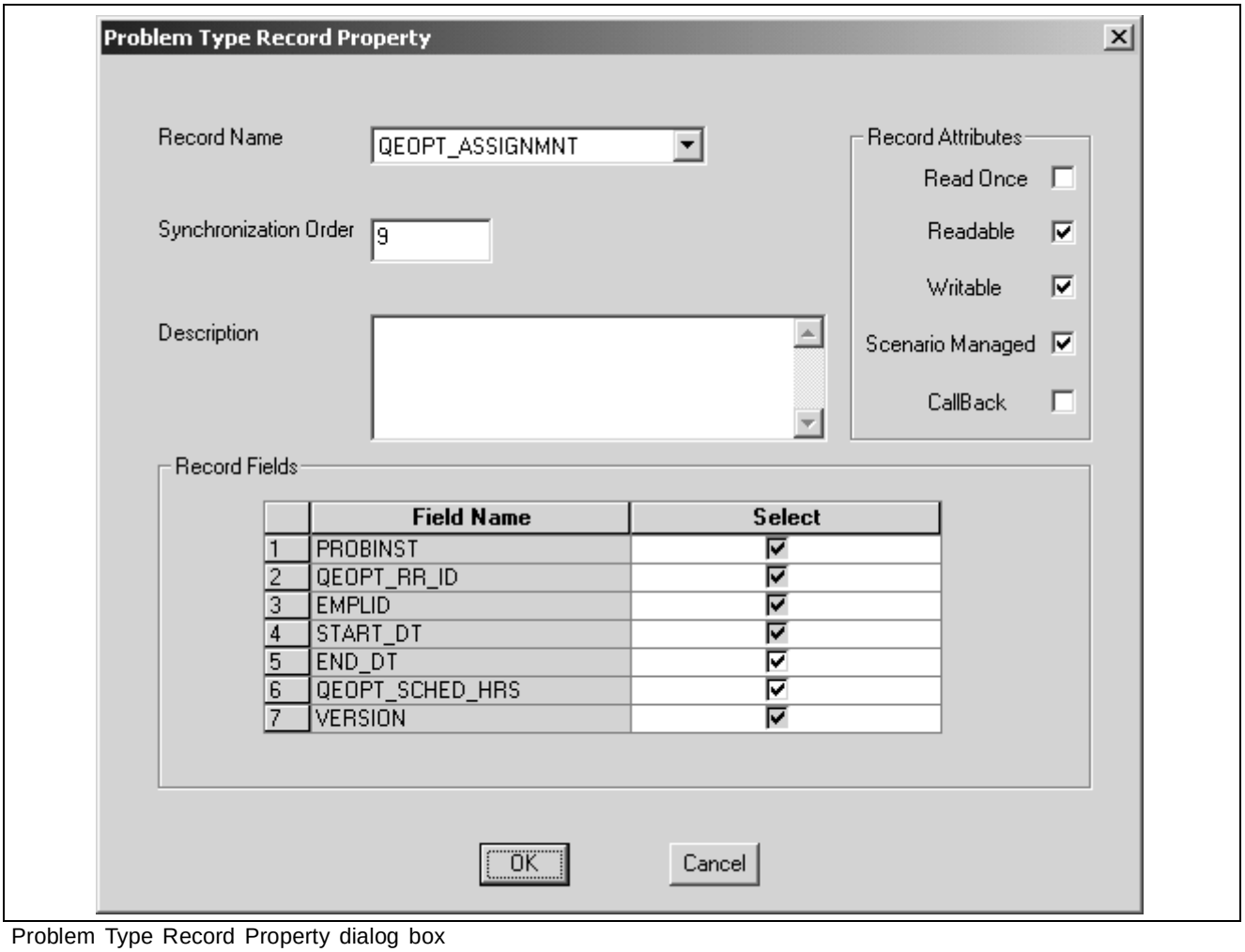
See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Optimization PeopleCode"

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer

Configuring Problem Type Records

In the problem type definition, select the Records tab, then select Insert, Record. The Problem Type Record Property dialog box appears.



Problem Type Record Property dialog box

Note. You can access the properties of an existing problem type record by right-clicking the record and selecting Problem Type Record Properties.

Record Name Select the record to use in the problem type definition.

Note. If you select a derived/work record, keep in mind that its scope in optimization PeopleCode is different than in other PeopleCode. When you use the **CreateOptEngine** function to instantiate an OptEngine object, each derived/work record is instantiated at level zero of the problem instance rowset. The record persists, and you can continuously modify its data across multiple transactions, until you shut down the optimization engine using the **ShutDown** method.

Synchronization Order Indicates the order in which the optimization engine reads the optimization application records. If a record has dependencies on another record, the dependent record should be read later. For example, the QE_RSM_EXERTGT record (synchronization order number is 4) depends on data in the QE_RO_MACH_CALS record (synchronization order number is 1). This order is determined by the application logic.

Read Once Select to have the record read only once during the initial load of the problem instance into the optimization engine. You cannot select Writeable if this check box is selected.

The optimization engine reads these records only once during the initial data load. It is assumed that the data in these records does not change (or the user doesn't care if it changes) from the initial load of the optimization engine until shutdown.

For the exercise machine problem, you might create a record that contains the name of an exercise machine and the number of calories one can burn on it. This information only needs to be read once by the optimization engine. Furthermore, the information will not change, so a VERSION field is not required.

Readable Select to have the record checked for updates by the optimization engine with every optimization transaction.

Readable records, besides being loaded during the initial load, are checked for updates by the optimization engine at the beginning of every optimization transaction. For every readable optimization application record, you must also create a corresponding optimization delete record and associate the readable record with the delete record. This process is explained later in this chapter.

For the exercise machine example, an appropriate readable record contains the name of a person who exercises, the start time and duration of the exercise, and the number of calories that the person wants to burn. This record is readable and scenario-managed. It has a VERSION field and a PROBINST field that contains the name of the person. Because this is pure source data, this data is not writable.

Writable Select to enable the optimization engine to modify rows for this record. A record can be both readable and writable. It's more common for records to be readable and writable instead of just writable.

A writable record contains result data from the optimization engine. For the exercise machine example, the system calculates this data every time you request an exercise summary. For this reason, it is purely writable.

Scenario-Managed Select to indicate that the record will contain data pertaining to multiple problem instances.

Note. Scenario-managed records must have a PROBINST key field.

See Chapter 3, "Designing Problem Type Definitions," Scenario Management, page 8.

Callback Select to enable the optimization engine to update its working data whenever this record changes.

Your problem type definition might include a record that you expect to change during the course of the optimization. If you want those changes to be taken into account by the optimization, you can define it as a callback record, so you can use provided PeopleCode callback methods to dynamically propagate those changes to the optimization's derived data structures. A callback record must be readable and writable.

Warning! If you select this check box for a record, you must ensure that you override all of the abstract callback placeholder methods that are defined in the extended PT_OPT_BASE:OptBase application class, even if it contains only a **Return** statement. Otherwise your Optimization PeopleCode plug-in will fail.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," OptBase Application Class.

Record Fields In the Record Fields list, select the fields in this record that need to be read into the optimization engine.

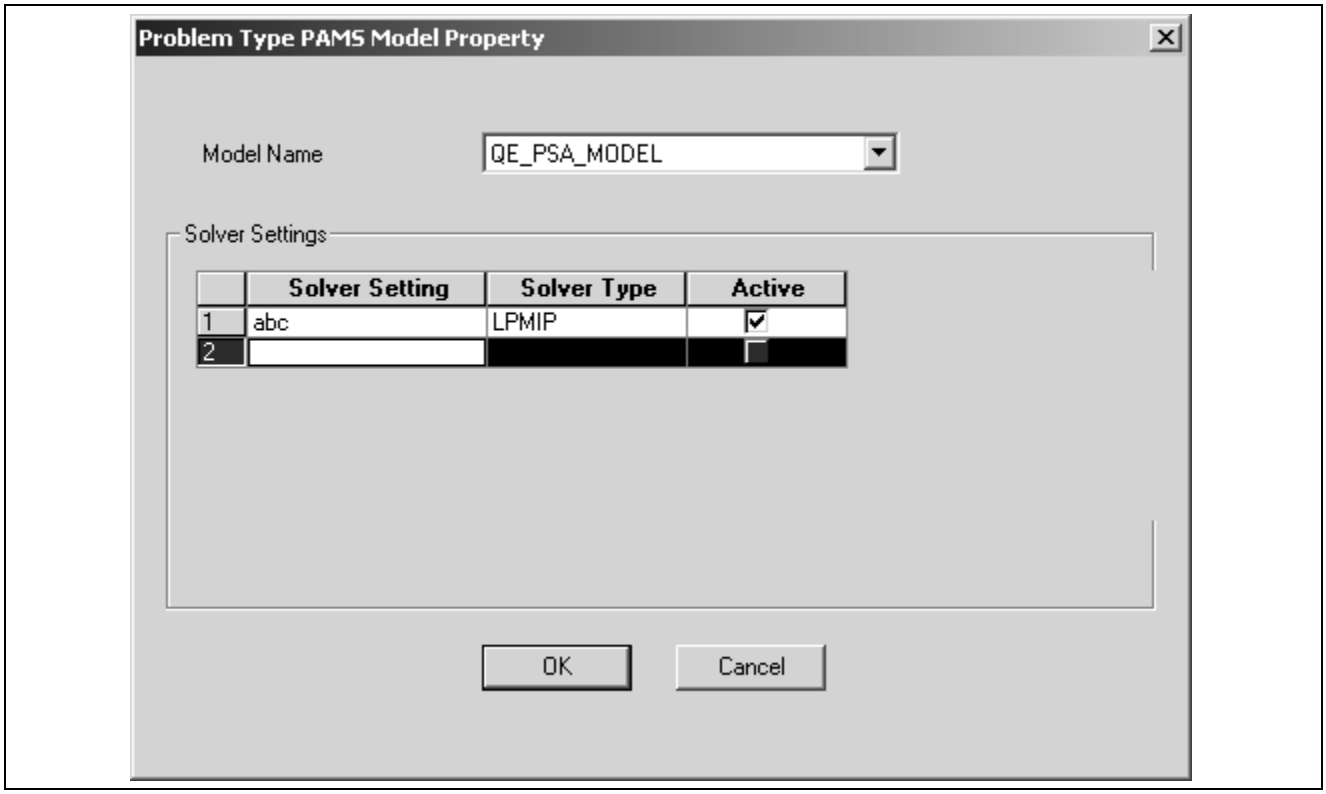
These are the fields that the OPI can access. Key fields and the VERSION field (if it exists) are always selected automatically. To conserve memory used by the optimization engine, select only the necessary fields.

Configuring Problem Type Models

You need to specify and configure problem type models only if both of the following are true:

- You selected PeopleCode Plugin in the problem type properties, indicating that your problem type definition should use the Optimization PeopleCode plug-in.
- Your application documentation indicates that an optimization model is necessary for the optimization application you're developing.

In the problem type definition, select the Models tab, then select Insert, PAMS Model. The Problem Type PAMS Model Property dialog box appears.



Problem Type PAMS Model Property dialog box

Note. Your application documentation discusses which models to specify, and what configuration settings to make for each model. You can access the properties of an existing problem type model by right-clicking the model and selecting Problem Type Model Properties.

Model Name Select the optimization model required to implement an optimization application with this problem type.

Solver Settings A solver setting is a collection of solver parameters with default values that define a particular solver behavior suitable for the optimization model. Specify one or more solver settings to make available to your optimization application, including:

- Solver Setting.

  Enter the name of the solver setting.

- Solver Type.

  Select the solver type: *LP* (linear programming), *MIP* (mixed integer programming), or *LPMIP* (both).

- Active.

  Select the active solver setting — only one solver setting can be active at a time.

Configuring Solver Parameters

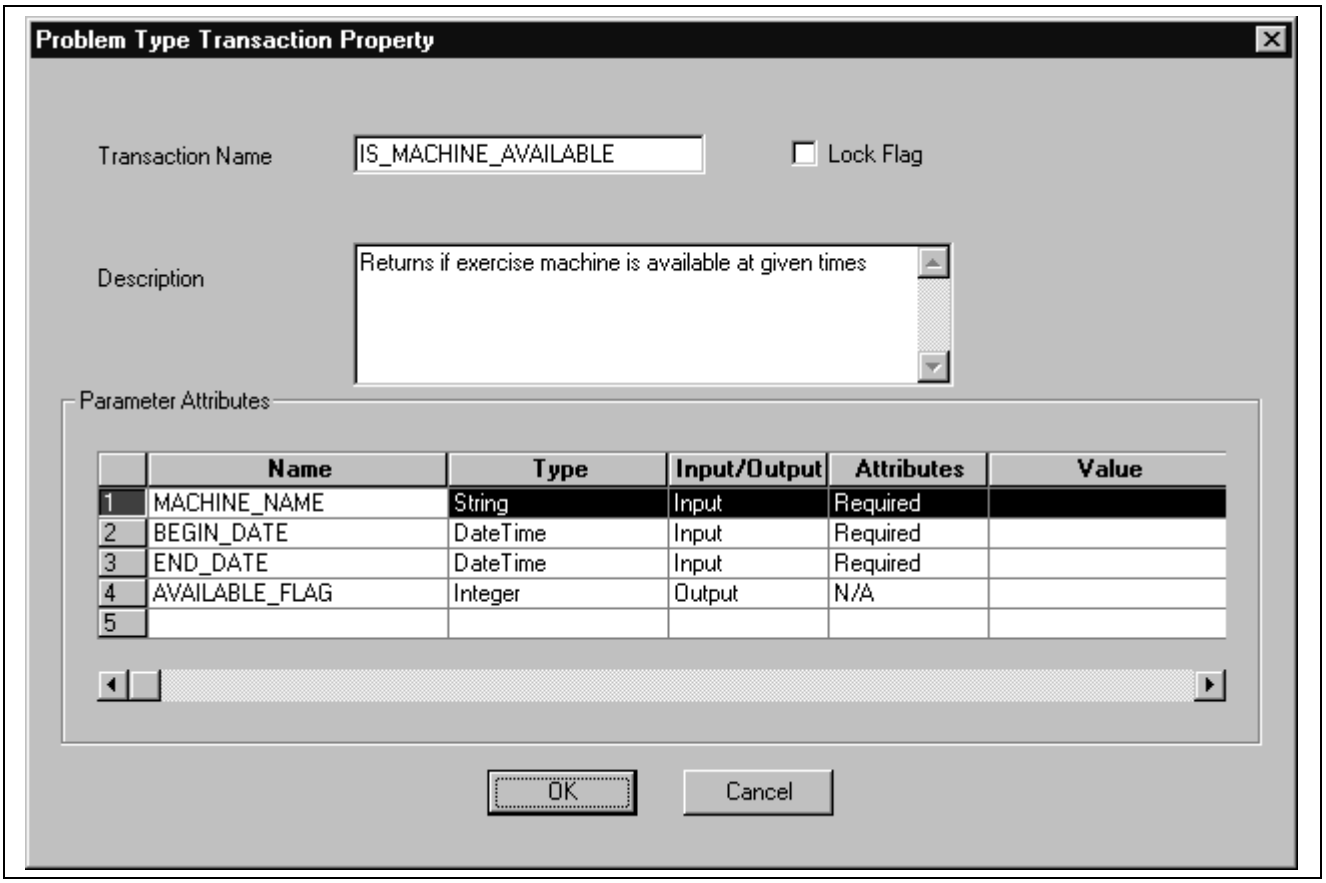
For each solver setting you specify, you can configure one or more *solver parameters.*

Double-click on a solver setting to access the Problem Type PAMS Solver Property dialog box. This dialog box has a grid with two columns: Parameter ID and Parameter Value.

Each solver type has a different set of available parameters, and each parameter has a default value. When you select a solver parameter from the Parameter ID dropdown list, its default value appears in the Parameter Value cell, and a new row appears for adding another parameter. Your application documentation discusses which parameters to specify for each solver setting, and what value to specify for each parameter.

Configuring Problem Type Transactions

In the problem type definition, select the Transactions tab, then select Insert, Transaction. The Problem Type Transaction Property dialog box appears.



Problem Type Transaction Property dialog box

Note. You can access the properties of an existing problem type transaction by right-clicking the transaction and selecting Problem Type Transaction Properties.

Transaction Name Enter the case-sensitive name of the transaction.

If the PeopleCode Plugin check box is selected in the problem type properties, this value must match the name of a method defined in the application class you specified for this problem type.

If the PeopleCode Plugin check box is *not* selected in the problem type properties, this value must match the name of a service defined in the OPI that you selected in the problem type properties.

The transaction name you specify must be distinct within a problem type.

For the exercise machine example, three transactions are needed. The QEOPT.DLL OPI implements these transactions:

- SOLVE solves the exercise machine problem.
- GET_SUMMARY produces a summary of exercises for a person.
- IS_MACHINE_AVAILABLE returns whether an exercise machine is available for a specified time.

The transaction name can contain up to 30 characters.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," OptBase Application Class.

Lock Flag Enter this flag to prevent changes to the optimization application tables while this transaction runs. Typically, this flag should be set for extremely fast but critical transactions where data integrity is crucial. In the exercise planning example, optimization transactions do not need the lock flag.

Important! The lock flag can hamper performance, so use it with caution.

Parameter Attributes

Each transaction can have any number of parameters.

If the application class method corresponding to this transaction has parameters, you must define a row in this grid with equivalent attributes for each of the parameters.

Name Enter the name of the parameter. The name must match the transaction parameter name defined in the OPI, or the equivalent method parameter defined in the application class you specified for this problem type.

The transaction parameter name can contain up to 20 characters, and it must be distinct within a problem type.

Type Select the parameter type (*String*, *Integer*, *Double*, *Date*, *DateTime*, *Time*, or arrays of these types, or *Record Array*). The type must match the transaction parameter type defined in the OPI, or the equivalent method parameter type defined in the application class you specified for this problem type.

Note. Do not pass an array of type Integer as a transaction parameter. Use an array of type Number instead.

Input/Output Select *Input*, *Output*, or *Both*.

Attributes Select *Required*, *Optional*, or *Default* (the parameter has a default value). This is not applicable to output parameters.

Note. If an input parameter is required, it must be supplied when you use either the RunSynch or RunAsynch PeopleCode methods.

Value If the Attributes field is set to *Default*, enter a default value for this parameter. If the type is *Record Array*, enter the name of the record. Otherwise, leave this blank.

Running the Optimization System Audit

After you have created the problem type definition, run SYSAUDIT with the optimization options selected. This ensures that the definition is valid and consistent.

To run the optimization system audit:

1. From the PeopleTools Utilities menu, select Audit, Perform System Audit.
2. Enter a run control ID.
3. On the System Audit page, select Audit Optimization Integrity, and click Run.
4. On the Process Scheduler Request page, ensure that the System Audit check box is selected, select a server name, and click OK.
5. When the System Audit page reappears, click Process Monitor (to the left of the Run button).
6. On the Process List page, at the end of the line for SYSAUDIT, click Details.
7. On the Process Detail page, click View Log/Trace.
8. On the View Log/Trace page, click the SYSAUDIT_XX file name.

   This file contains the audit report for your optimization.

See Also

Enterprise PeopleTools 8.45 PeopleBook: Data Management, "Ensuring Data Integrity," Running SYSAUDIT

Changing Existing Problem Type Definitions

This section discusses how to change:

- Optimization application records.
- Optimization transactions.

Changing Optimization Application Records

To change optimization application records in a problem type definition:

1. Shut down all the running optimization engines that use this problem type definition.
2. Shut down other optimization engines if record definitions are being shared by other problem type definitions.
3. Delete all existing problem instances using the DeleteOptProbInst PeopleCode function.

   See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," DeleteOptProbInst.
4. Empty the optimization application tables.
5. Make record definition changes and build the records in PeopleSoft Application Designer.

   See Chapter 3, "Designing Problem Type Definitions," Creating and Building Optimization Records, page 9.

6. Open the problem type in PeopleSoft Application Designer, insert any new records or make appropriate changes to reflect changed record definitions, and save the problem type.

   Run SYSAUDIT with the optimization options selected.

   Skip the steps about inserting transactions.

7. Change the OPI to reflect the changes to optimization application records.

   If the records do not match the plug-in, the program will fail.

8. Call the InsertOptProbInst PeopleCode function to recreate problem instances.

   See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," InsertOptProbInst.

Changing Optimization Transactions

To change optimization transactions in a problem type definition:

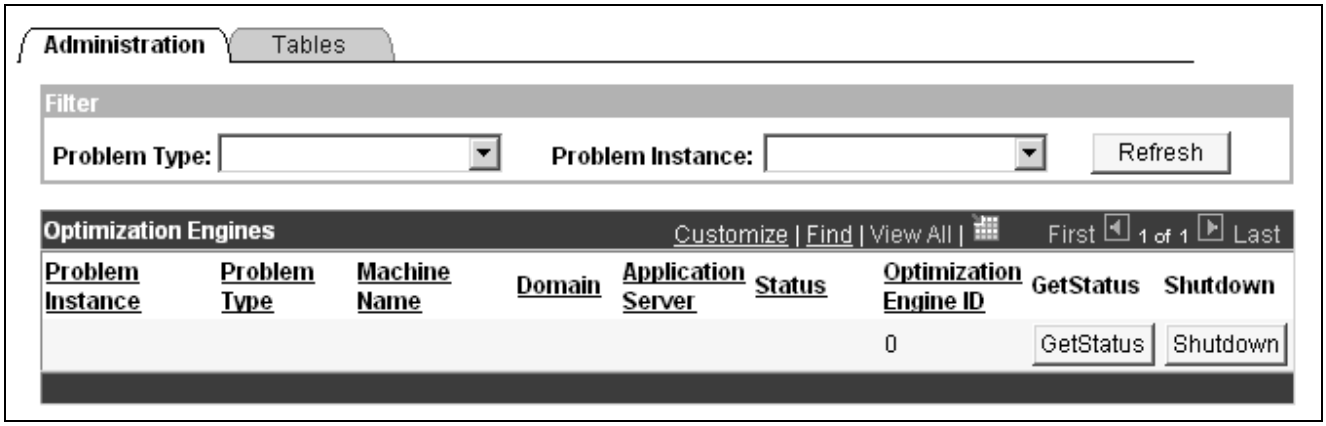
1. Shut down all the running optimization engines that use the problem type definition.
2. Open the problem type definition in PeopleSoft Application Designer, insert any new transactions or make appropriate changes to existing ones, and save the problem type.

   Skip the steps about inserting records.

3. Change the OPI to reflect the changes to optimization transactions.
4. Change optimization PeopleCode to reflect the changes (add, remove, and update parameters).

Administering Optimization Engines

In PeopleSoft Pure Internet Architecture, select PeopleTools, Utilities, Optimization, Administer Engines, Administration to access the Administer Engines - Administration page.



Administer Engines - Administration page

From this page, an administrator can view all the optimization engines that are loaded with a problem instance, get their current status, and shut them down if necessary. If Integration Broker messaging is enabled, this page enables an administrator to centrally control optimization engines in all domains connected to the same PeopleSoft application database.

Note. This page uses features of application messaging to show status and allow shutdown; you must configure Integration Broker for basic messaging in order to use these features. Without Integration Broker, the page still displays a list of the optimization engines (click Refresh to populate it), but the status and shutdown functionality is disabled.

The following fields are displayed on the Administration page.

Problem Type Specify a problem type to show only the optimization engines that are loaded with problem instances associated with this problem type. A problem type can have many problem instances that are loaded in different optimization engines.

Problem Instance Specify a problem instance to show only the optimization engines that are loaded with this problem instance. Multiple optimization engines can load the same problem instance as long as they are associated with distinct domains.

Refresh Click to populate the page with the optimization engine status information.

Machine Name Displays the name of the machine upon which the optimization engine exists.

Domain Displays the name of the application server or process scheduler domain under which the optimization engine exists.

Application Server If *Yes*, the optimization engine exists in an application server domain. If *No*, the optimization engine exists in a process scheduler domain. This is determined by querying to see if the string *PRCS* exists in the full path of the domain under which the optimization engine is running.

Status Displays the status of the optimization engine:

- *OptEngine Shutdown.*
- *OptEngine is Busy Loading.*
- *OptEngine is Idle.*
- *OptEngine is Busy.*
- *OptEngine must Reboot.*
- *OptEngine Unknown.*

Optimization Engine ID Displays an ID number that maps the problem instance to an optimization engine instance.

GetStatus Click to display a status message for a specific optimization engine.

Note. This button works only if Integration Broker is configured for basic messaging, or if the optimization engine is in the current application server domain.

Shutdown Click to shut down a specific optimization engine.

The page still displays the optimization engine's information, with a status of *OptEngine Shutdown.* Click Refresh to remove that row from the page.

Note. This button works only if Integration Broker is configured for basic messaging, or if the optimization engine is in the current application server domain.

Setting Up Integration Broker

Before you can use lights-out mode and other optimization features, you must first configure PeopleSoft Integration Broker for basic messaging.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker*.

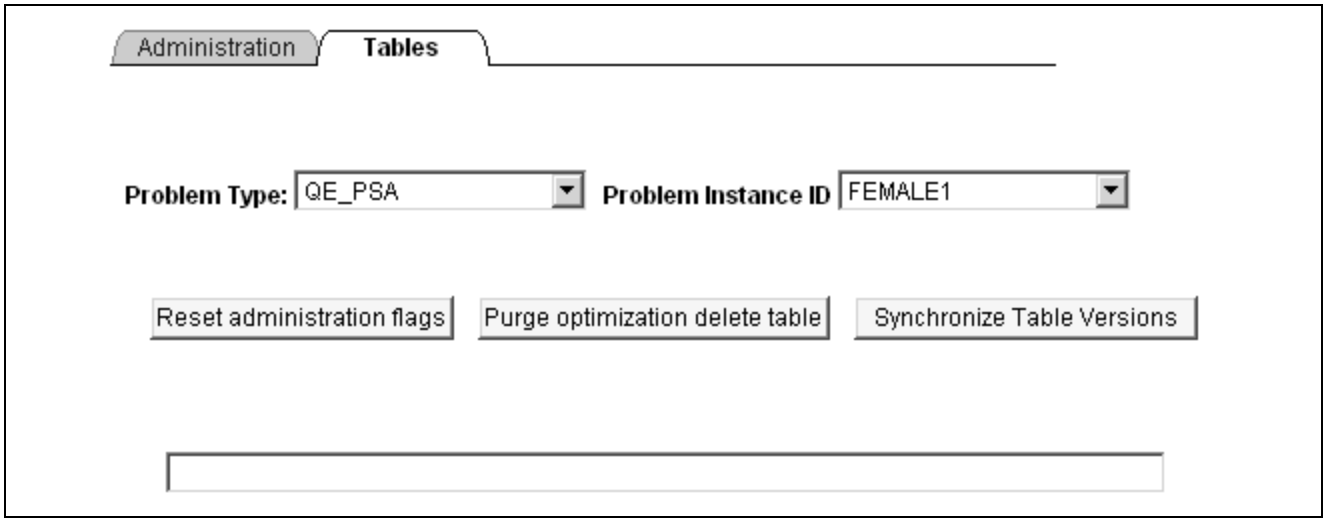
The only PeopleSoft Integration Broker elements specific to optimization engine administration are two transactions delivered with your PeopleSoft application. One transaction is type *InSync*, the other is type *OutSync*, and both use the OPT_CALL message. Ensure that they're both active on the Transactions page of the default local node definition.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker*, "Configuring Nodes and Transactions".

Administering Optimization Tables

Access the Administer Engines - Tables page.

Note. Shut down all running optimization engines before using this page.



Administer Engines - Tables page

Problem Type Select the problem type for which you want to administer optimization tables.

Problem Instance ID (Optional) Select a problem instance for which to administer optimization tables.

Reset administration flags Click to clear the administration flags for the problem type.

Note. When you shut down or restart an optimization engine, the administration flags are cleared automatically.

Purge optimization delete tables	Click to clear the optimization delete tables for the problem type or problem instance.
Synchronize Table Versions	When you use Datamover to move data from one database to another, it is quite often the case that the versions of problem instance data and the PSOPTSYNC table are out of synchronization. Click this button to synchronize the PSOPTSYNC table with the optimization tables.

Updating Solver Licenses

Use the Administer license page to update a solver software license. PeopleSoft Optimization Framework uses third-party solver software. In some cases, the solver software is activated by a license.

Note. Currently, no optimization application requires updating the solver license. You should update solver licenses only on instructions from PeopleSoft.

To update solver licenses:

1. In a browser, select PeopleTools, Utilities, Optimization, Solver Licenses.
2. Enter an optimization solver type, such as LP/MIP.

   The optimization engine identifies the third-party solver software by its solver type.
3. On the Administer license page, enter the new license code in the Encrypted License Code field.

CHAPTER 4

Optimization PeopleCode

This chapter discusses how to:

- Use optimization PeopleCode on the application server.
- Use optimization PeopleCode in an Application Engine program.
- Perform optimization in PeopleCode.
- Use lights-out mode with optimization.
- Use optimization built-in functions.
- Use OptEngine class methods.
- Use OptEngine class properties.
- Use the OptBase application class.
- Use OptBase class methods.
- Use OptInterface class methods.

Using Optimization PeopleCode on the Application Server

While running optimization PeopleCode on the application server, ensure that changed data is committed to the database before calling the CreateOptEngine optimization function and the following OptEngine class methods:

- RunSynch
- RunAsynch
- CheckOptEngineStatus
- ShutDown
- SetTraceLevel
- GetTraceLevel
- InsertOptProbInst
- DeleteOptProbInst

Note. The PeopleCode functions CommitWork and DoSaveNow can be called within a step to save uncommitted data to the database before calling the listed functions and methods. Keep in mind that forcing a commit on pending database updates is a serious step; it prevents roll-back on error. CreateOptEngine, ShutDown, InsertOptProbInst and DeleteOptProbInst modify the database, so take care when terminating the Application Engine program without committing the changes made by those calls.

Using Optimization PeopleCode in an Application Engine Program

When you write an optimization PeopleCode program in an Application Engine program and you schedule it in PeopleSoft Process Scheduler, you must set the process definition with a process type of *Optimization Engine*. Other process types do not allow optimization PeopleCode in Application Engine programs.

While using optimization PeopleCode in Application Engine programs, make sure data is committed before calling the CreateOptEngine optimization function and the following OptEngine class methods:

- RunSynch
- RunAsynch
- CheckOptEngineStatus
- ShutDown
- SetTraceLevel
- GetTraceLevel
- InsertOptProbInst
- DeleteOptProbInst

Note. You can call the PeopleCode functions CommitWork and DoSaveNow within a step to save uncommitted data to the database before calling the listed functions and class methods. Keep in mind that forcing a commit on pending database updates is a serious step; it prevents roll-back on error. CreateOptEngine, ShutDown, InsertOptProbInst and DeleteOptProbInst modify the database, so take care when terminating the Application Engine program without committing the changes made by those calls.

Performing Optimization in PeopleCode

This section discusses how to:

- Create new problem instances.
- Load problem instances into an optimization engine.
- Run optimization transactions.
- Invoke the Optimization PeopleCode plug-in.
- Shut down optimization engines.
- Delete existing problem instances.
- Program for database updates.

Creating New Problem Instances

To create a new problem instance for a problem type:

1. Call the function InsertOptProbInst with the problem type and problem instance as arguments to create the problem instance ID in PeopleTools metadata.
2. Using PeopleSoft Application Engine or a similar mechanism, load the optimization application tables with data.

Use the problem instance ID as the key value in scenario-managed optimization application tables.

The problem instance is now ready to be loaded into an optimization engine.

Note. You can load multiple copies of the same problem instance into multiple instances of an optimization engine, provided that each instance of the optimization engine resides in a different application server domain. Each problem instance loaded into a given domain must be unique. Within a given domain, you cannot have the same problem instance in more than one optimization engine. The optimization engine maintains data integrity by checking to see if the data has been altered by another user (refer to the steps in the optimization system architecture description). Try to maintain data consistency when the same problem instance uses the same database in different domains.

See Also

Chapter 2, "Understanding PeopleSoft Optimization Framework," PeopleSoft Optimization Framework System Architecture, page 4

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Optimization PeopleCode," InsertOptProbInst

Loading Problem Instances Into an Optimization Engine

Call the CreateOptEngine function to load an optimization engine with a problem instance. It takes problem instance ID and a mode parameter with %Synch and %Asynch as possible values and returns a PeopleCode object of type OptEngine.

You can run the PeopleCode on the application server or from PeopleSoft Application Engine.

Loading Problem Instances by Running PeopleCode on the Application Server

To block PeopleCode from running on the application server until the load is done (synchronous mode), use the %Synch value for the mode parameter. An error is generated if the load isn't successful. The application server imposes a timeout beyond which the PeopleCode and optimization engine load are terminated. Here is a code example:

```
Local OptEngine &myopt;
&myopt = CreateOptEngine("PATSMITH", %Synch);
```

To load the optimization engine without blocking the PeopleCode from running (asynchronous mode) on the application server, use the %Asynch value for the mode parameter. The optimization engine performs a preliminary check of the load request and returns the OptEngine object if it is successful or an error if it is unsuccessful. A successful return does not mean that the load was successful. You must then use repeated CheckOptEngineStatus methods on the returned OptEngine object to determine whether the optimization engine is done with the load and whether it was successful. Here is a code example:

```
Local OptEngine &myopt;
&myopt = CreateOptEngine("PATSMITH", %Asynch);
```

Loading Problem Instances by Running PeopleCode in PeopleSoft Application Engine

Both synchronous (%Synch) and asynchronous (%Asynch) modes block the PeopleCode from running on PeopleSoft Application Engine until the load is done. Use only %Asynch while loading an optimization engine.

The absolute number of optimization engine instances that may be loaded in a given domain is governed by a configuration file loaded by Tuxedo during its domain startup.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Optimization PeopleCode," CheckOptEngineStatus

Chapter 5, "Administering Optimization Server Components," page 101

Running Optimization Transactions

You send an optimization transaction to the optimization engine using the RunSynch and RunAsynch methods. Both are methods on an OptEngine object. The OptEngine object can be created either by calling CreateOptEngine (if the optimization engine is not loaded already) or by calling GetOptEngine (if the optimization engine is already loaded). Both RunSynch and RunAsynch have the same signature, except that RunSynch runs the optimization transaction in synchronous mode and RunAsynch runs it in asynchronous mode. Both return an integer status code. You can run transactions either on the application server or with PeopleSoft Application Engine.

To invoke an optimization transaction:

1. Use the GetOptEngine function to get the OptEngine object as a handle for the optimization engine that has loaded a problem instance ID.

   Use the CreateOptEngine function to create the OptEngine object for a new optimization engine if the problem instance has not been loaded.

2. Call RunSynch or RunAsynch to send an optimization transaction to the optimization engine to be run in synchronous or asynchronous mode.

3. If the transaction is run in synchronous mode (RunSynch), use the OptEngine methods GetString, GetNumber, and so on, to retrieve the output result from the optimization transaction.

   The transaction names, parameter names, and data types are viewable in the problem type in PeopleSoft Application Designer.

4. If the transaction is run in asynchronous mode, use the OptEngine method CheckOptEngineStatus to check the status of the optimization transaction in the optimization engine.

   After the transaction is done, result data is available in the database for retrieval using standard PeopleCode mechanisms.

Running Optimization Transactions from the Application Server

To block the PeopleCode from running on the application server until the optimization transaction is done (synchronous mode) and receives the results, use RunSynch to send an optimization transaction. An error status code is returned if the transaction isn't successful. If successful, you can use other methods to retrieve the results from the transaction call. The application server imposes a timeout beyond which the PeopleCode and optimization engine transaction are terminated.

To run a transaction without blocking PeopleCode from running (asynchronous mode) on the application server, use RunAsynch to send an optimization transaction. In this mode, the optimization engine performs a preliminary check of the transaction request and returns a success or failure status code. A successful return does not mean that the transaction is successful; it means only that the syntax is correct. You must then use repeated calls to the CheckOptEngineStatus method on the OptEngine object to determine whether the optimization engine is done with the transaction and whether it is successful.

RunAsynch does not allow transaction output results to be returned. Use this method for long-running transactions that return results entirely through the database.

Running Optimization Transactions from PeopleSoft Application Engine

Both synchronous (RunSynch) and asynchronous (RunAsynch) methods block the PeopleCode from running on PeopleSoft Application Engine until the optimization transaction is done. RunSynch allows output results to be returned, but it should be used for transactions that are fast (less than 10 seconds). RunAsynch does not have a time limit, but it does not return output results.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Optimization PeopleCode," RunAsynch

Invoking the Optimization PeopleCode Plug-In

If you're developing an optimization application that uses the Optimization PeopleCode plug-in, you must do the following to invoke the plug-in:

- Develop a PeopleCode application class that extends the PT_OPT_BASE:OptBase class.
- Define methods in your application class that use the PeopleCode OptInterface class to perform your optimization functions.
- Define a problem type that specifies the Optimization PeopleCode plug-in, by selecting the PeopleCode Plugin check box in the problem type properties.
- Also in the problem type properties, specify the application package and application class that you developed.
- Define transactions in your problem type definition that correspond to the methods you developed in your application class, with corresponding parameters.

See Also

Chapter 3, "Designing Problem Type Definitions," Creating Problem Type Definitions, page 10

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Optimization PeopleCode," CreateOptInterface

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Optimization PeopleCode," OptBase Application Class

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Optimization PeopleCode," OptInterface Class Methods

Shutting Down Optimization Engines

Use the GetOptEngine function to get the OptEngine object as a handle for the optimization engine that loaded a problem instance ID.

Call the OptEngine method named ShutDown to shut down the optimization engine. This ends the optimization engine process with the current problem instance ID. Based on application server settings, the system restarts a new, unloaded optimization engine process that can be loaded with any other problem instance.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Optimization PeopleCode," ShutDown

Deleting Existing Problem Instances

To delete an existing problem instance for a problem type:

1. Shut down any optimization engines that have this problem instance currently loaded.
2. Using PeopleSoft Application Engine or a similar mechanism, delete the data in the optimization application tables pertaining to that problem instance.

   Use the problem instance ID as the key value to find and delete problem instance rows from scenario-managed optimization application tables.
3. Call the function DeleteOptProbInst with the problem type and problem instance as arguments to delete the problem instance ID from PeopleTools metadata.

Note. If you attempt to delete an existing problem instance that is loaded in a running optimization engine, DeleteOptProbInst returns %OptEng_Fail, and the optional status reference parameter is set to %OptEng_Exists.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Optimization PeopleCode," DeleteOptProbInst

Programming for Database Updates

You must plan for uncommitted database changes in your optimization PeopleCode. The Optimization Framework detects pending database updates, and generates a failure status if data is not committed to the database before certain optimization methods are called.

This checking for database updates happens in runtime for the CreateOptEngine function and the following methods: RunSync, RunAsync, Shutdown, GetTraceLevel, and SetTraceLevel. Ensure that your PeopleCode performs proper database updates and commits before you execute these methods. If you use the optional parameter for detailed status that is available for these functions, or the DetailedStatus property that is available for the methods, you can check for the status of %OptEng_DB_Updates_Pending to see if there is a pending database update.

Note. The pending database update may have happened considerably earlier in the code. Forcing a commit within your PeopleCode to avoid this problem prevents roll-back on database error. Forcing a commit should be used with care.

The InsertOptProbInst and DeleteOptProbInst functions can be called only inside FieldChange, PreSave and PostSave PeopleCode events, and in Application Message Subscription PeopleCode and Workflow.

This database update checking happens in compile time for the following functions: InsertOptProbInst and DeleteOptProbInst. Make sure that there are no pending database updates before you execute these methods.

Using Lights-Out Mode with Optimization

This section provides an overview of lights-out mode, and discusses how to:

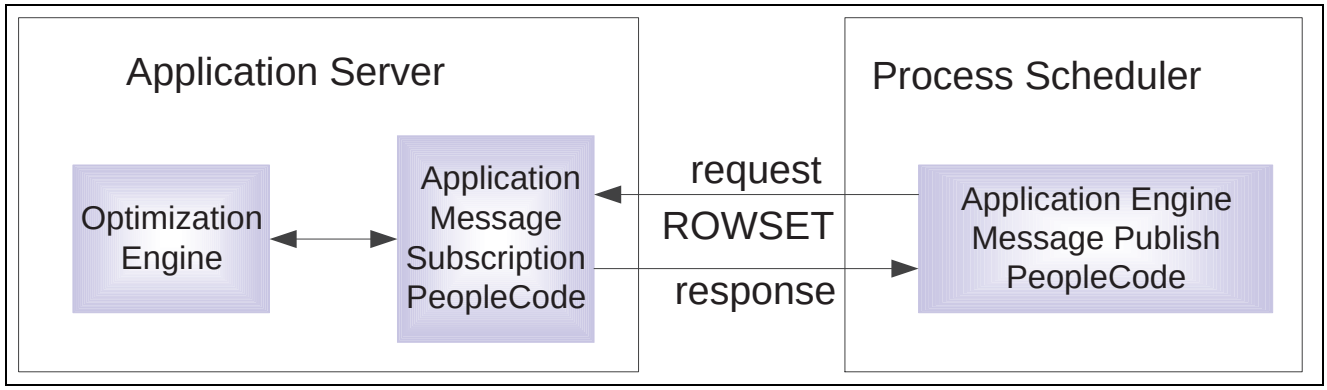
- Create a request message.
- Create a response message.
- Edit the request PeopleCode.
- Edit the response PeopleCode.

Understanding Lights-out Mode

Some optimization applications can take several hours to run. These are typically run as overnight batch jobs every night when the work load is small to regenerate the optimization solution and have it ready for end users to use in the morning — hence the term *lights-out mode.*

In the current release, application messages communicate between the Application Engine batch job and the online optimization engine. After the Application Engine job completes and the optimization solution has been written to the database, an application message initiates the download of the data from the database batch job to the online optimization engine.

Lights-out mode uses an Application Engine PeopleCode program within PeopleSoft Process Scheduler to send requests to an application server and receive responses from it. Within the application server, the OnRequest PeopleCode runs an optimization engine process.



Lights-out process

This request and response is in the form of a rowset as shown by the example supplied with optimization, the OPT_CALL message. Also supplied as an example is an Application Engine message publish PeopleCode program called PT_OPTCALL.

Important! PeopleSoft Application Engine includes an action of type *Log Message*, which PeopleSoft Process Scheduler uses to record its activity in the PS_MESSAGE_LOG table. The PeopleCode MessageBox and WinMessage built-in functions also record their activity in the PS_MESSAGE_LOG table.

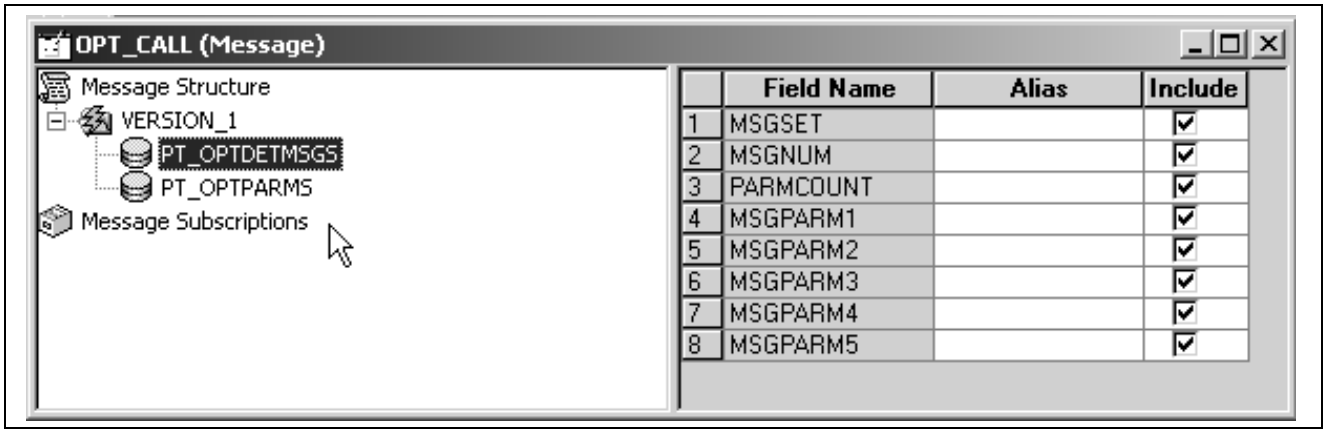
During lights-out optimization, these processes can conflict with each other or with the optimization engine when one process locks a row of the table, and another process tries to access the same row.

To prevent this conflict, pay close attention to where the MessageBox or WinMessage built-in functions are used in your Application Engine PeopleCode. In general, there can't be any outstanding database updates pending when communicating with the optimization engine using application messages.

The OPT_CALL Message

The OPT_CALL message is an example of what the lights-out process uses as the message for optimization. The OPT_CALL message has a structure using a record, PT_OPTPARMS, having the fields PARMKEY and VALUE which represent a name/value pair. These send requests and responses from the Application Engine PeopleCode in PeopleSoft Process Scheduler to and from the message OnRequest PeopleCode in the application server.

The OPT_CALL message also uses a record, PT_OPTDETMSGS, which contains the information needed for processing a detailed message.



OPT_CALL message structure

Creating a Request Message

This section provides an overview of the request message and describes how to create messages that:

- Create an optimization engine.
- Check optimization engine status.
- Run an optimization engine transaction.
- Set the trace level.
- Get the trace level.
- Shutdown an optimization engine.

Understanding the Request Message

For optimization, the Application Engine PeopleCode in PeopleSoft Process Scheduler sends a request OPT_CALL message. The message uses rowsets built from PT_OPTPARMS records as the request. You can use the following rowset structures as an example of how to perform certain optimization actions, by sending them as requests from the application engine program in the process scheduler to the message subscription PeopleCode in the application server.

Creating an Optimization Engine

To create an optimization engine, structure the rowset as follows, using the PT_OPTPARMS record. You set key values using the PARMKEY field, and then set a value for that key field in the VALUE field.

PARMKEY Field	VALUE Field
OPTCMD	CREATE Causes the OnRequest PeopleCode to load an optimization engine. The OPT_CALL example executes the CreateOptEngine function.
PROBINST	The name of the problem instance.

PARMKEY Field	VALUE Field
PROCINSTANCE	The name of the process instance for this process scheduler job.
SYNCH	Y if this optimization engine load is to occur synchronously, N if asynchronously.

Checking Optimization Engine Status

To check optimization engine status (for example, to see when it finishes loading), structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	VALUE Field
OPTCMD	CHECK_STATUS Causes the OnRequest PeopleCode to check the status of an optimization engine. The OPT_CALL example executes the CheckOptEngineStatus function.
PROBINST	The name of the problem instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.

Running a Transaction

To run a transaction, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	VALUE Field
OPTCMD	RUN Causes the OnRequest PeopleCode to run an optimization transaction. The OPT_CALL example executes the GetOptEngine method and either the RunSynch or RunAsynch method.
PROBINST	The name of the problem instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.
SYNCH	Y for a synchronous transaction, N for asynchronous.
TRANSACTION	The name of the transaction to run.
The names of one or more transaction parameters.	The value of each named transaction parameter.

Setting the Trace Level

To set a trace level, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	VALUE Field
OPTCMD	SET_TRACE_LEVEL Causes the OnRequest PeopleCode to set the severity level at which events will be logged for an optimization engine. The OPT_CALL example executes the SetTraceLevel method.
PROBINST	The name of the problem instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.
COMPONENT	One of the following values: • %Opt_Engine — server activity of the running optimization engine. • %Opt_Utility — low level elements that support the running optimization engine. • %Opt_Datacache — the in-memory database cache. • %Opt_Plugin — the plugin being used for the running optimization engine.
SEVERITY_LEVEL	The severity level to log. The following list starts with the most severe level; the level you specify includes all higher levels. For example, if you specify %Severity_Error, it logs %Severity_Fatal, %Severity_Status, and %Severity_Error messages and filters out the others. • %Severity_Fatal • %Severity_Status • %Severity_Error • %Severity_Warn • %Severity_Info • %Severity_Trace1 • %Severity_Trace2

Getting the Trace Level

To get a trace level, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	VALUE Field
OPTCMD	GET_TRACE_LEVEL Causes the OnRequest PeopleCode to get the severity level at which events will be logged for an optimization engine. The OPT_CALL example executes the GetTraceLevel method.
PROBINST	Set to the name of the problem instance.

PARMKEY Field	VALUE Field
PROCINSTANCE	Set to the name of the process instance for this process scheduler job.
COMPONENT	One of the following values: • %Opt_Engine — server activity of the running optimization engine. • %Opt_Utility — low level elements that support the running optimization engine. • %Opt_Datacache — the in-memory database cache. • %Opt_Plugin — the plugin being used for the current opt engine.

Shutting Down an Optimization Engine

To shut down an optimization engine, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	VALUE Field
OPTCMD	SHUTDOWN Causes the OnRequest PeopleCode to shut down an optimization engine. The OPT_CALL example executes the Shutdown method.
PROBINST	The name of the problem instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.

Creating a Response Message

This section provides an overview of the response message and describes how to create messages that:

- Send optimization status.
- Send a detailed message.

Understanding the Response Message

For optimization, the message PeopleCode in application server receives the request messages, performs an optimization actions, and sends response OPT_CALL messages. One message uses rowsets built from PT_OPTPARMS records, the other uses rowsets from PT_DETMSGS records. You can use the following rowset structures as an example of how to send responses from the message subscription PeopleCode in the application server to the application engine program in the process scheduler.

Sending Optimization Status

To send the status of the optimization functions and methods called within the message OnRequest PeopleCode, structure the rowset as follows using the PT_OPTPARMS record. The optimization functions and messages are called in response to the request input message. You set key values using the PARMKEY field, and then set a value for that key field in the VALUE field.

PARMKEY Field	VALUE Field
STATUS	The return status of the optimization function or method that is called in the message PeopleCode.
DETAILED_STATUS	The optional detailed status returned by many of the optimization functions and methods.

Sending a Detailed Message

To send a detailed message, structure the rowset as follows, using the PT_DETMSGS record. You set key values using the PARMKEY field, and then set a value for that key field in the VALUE field.

PARMKEY Field	VALUE Field
MSGSET	The message set number. In the case of optimization, the message set number is 148.
MSGNUM	The name of the detailed message.
PARMCOUNT	The number of message parameters for the detailed message. There can be up to five parameters.
MSGPARM1	The first parameter value.
MSGPARM2	The second parameter value.
MSGPARM3	The third parameter value.
MSGPARM4	The fourth parameter value.
MSGPARM5	The fifth parameter value.

Editing the Request PeopleCode

The PT_OPTCALL Application Engine program serves as a template. It is delivered with all the sections marked as inactive. You can edit the program to suit your needs, then mark the appropriate sections active before running it. You can also use the program as a guide to creating your own Application Engine program.

The program uses these steps to send request messages to perform the following tasks:

1. Load the optimization engine.
2. Wait for the optimization engine load to finish.
3. Run an optimization transaction against the loaded optimization engine.
4. Wait for the optimization transaction to finish running.
5. Set the trace level.
6. Get the trace level.
7. Shut down the optimization engine.

You can edit steps 1 and 3 to run an optimization transaction. You can also use the entire program as a template to create your own Application Engine program.

Loading an Optimization Engine

In step 1, enter the name of your problem instance. In this example, the name of the problem instance is FEMALE1.

If you have multiple domains, enter the local node name and the machine name and port number for your application server. In this case, the local node name is %LocalNode and the machine name and port number are foo111111:9000.

```
Local Message &MSG;
Local Message &response;

Component string &probid;
Component string &isSync;
Component string &procinst;
Local integer &nInst;
Local string &url;

Local Rowset &rs;
Local Row &row;
Local Record &rec;

Local string &stName;
Local integer &stVal;

&MSG = CreateMessage(Message.OPT_CALL);
&rs = &MSG.GetRowset();

&row = &rs.GetRow(1);
&rec = &row.GetRecord(Record.PT_OPTPARMS);
&rec.PARMKEY.Value = "OPTCMD";
&rec.VALUE.Value = "CREATE";

&rs.InsertRow(1);
&rec = &rs.GetRow(2).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROBINST";
&rec.VALUE.Value = "FEMALE1";
&probid = "FEMALE1";

&rs.InsertRow(2);
&rec = &rs.GetRow(3).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROCINSTANCE";
&nInst = Record.PT_OPT_AET.PROCESS_INSTANCE.Value;
&rec.VALUE.Value = String(&nInst);
&procinst = String(&nInst);

&rs.InsertRow(3);
&rec = &rs.GetRow(4).PT_OPTPARMS;
&rec.PARMKEY.Value = "SYNCH";
&rec.VALUE.Value = "N";
&isSync = "N";
```

```
/* Specify the Application Server domain URL (foo111111:9000 in this example)
*/
&response = &MSG.SyncRequest(%LocalNode, "//foo111111:9000 e");

If &response.ResponseStatus = 0 Then
   &stName = &response.GetRowset().GetRow(1).GetRecord(Record.PT_OPTPARMS).Get⇒
Field(Field.PARMKEY).Value;
   &stVal = Value(&response.GetRowset().GetRow(1).GetRecord(Record.PT_⇒
OPTPARMS).GetField(Field.VALUE).Value);
   If &stName = "STATUS" And
         &stVal = %OptEng_Fail Then
      /* Check detailed message here */
      throw CreateException(148, 2, "Can not send to OptEngine");
   End-If;
End-If;
```

Running An Optimization Transaction

In step 3, enter the name of your optimization transaction and its parameter name/value pairs. In this example, the transaction name is TEST_LONG_TRANS, the first parameter name/value pair is Delay_in_Secs and 30, and the second parameter name/value pair is Sleep0_Work1 and 0.

The parameter values are stored as strings. You may need to convert them in the OnRequest PeopleCode.

```
Local Message &MSG;
Local Message &response;

Local Rowset &rs, &respRS;
Local Row &row;
Local Record &rec, &msgRec;

Component string &probid;
Component string &procinst;
Component string &isSync;
Local string &url = "";
Local integer &parmCount, &msgSet, &msgNum;

&MSG = CreateMessage(Message.OPT_CALL);
&rs = &MSG.GetRowset();

&row = &rs.GetRow(1);
&rec = &row.GetRecord(Record.PT_OPTPARMS);
&rec.PARMKEY.Value = "OPTCMD";
&rec.VALUE.Value = "RUN";

&rs.InsertRow(1);
&rec = &rs.GetRow(2).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROBINST";
&rec.VALUE.Value = &probid;
```

```
&rs.InsertRow(2);
&rec = &rs.GetRow(3).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROCINSTANCE";
&rec.VALUE.Value = &procinst;

&rs.InsertRow(3);
&rec = &rs.GetRow(4).PT_OPTPARMS;
&rec.PARMKEY.Value = "SYNCH";
&rec.VALUE.Value = &isSync;

&rs.InsertRow(4);
&rec = &rs.GetRow(5).PT_OPTPARMS;
&rec.PARMKEY.Value = "TRANSACTION";
&rec.VALUE.Value = "TEST_LONG_TRANS";

&rs.InsertRow(5);
&rec = &rs.GetRow(6).PT_OPTPARMS;
&rec.PARMKEY.Value = "Delay_in_Secs";
&rec.VALUE.Value = "30";

&rs.InsertRow(6);
&rec = &rs.GetRow(7).PT_OPTPARMS;
&rec.PARMKEY.Value = "Sleep0_Work1";
&rec.VALUE.Value = "0";

/* SyncRequest will carry a url */
SQLExec("select URL from PSOPTSTATUS where PROBINST=:1 AND URL NOT LIKE '%:0';",⇒
 &probid, &url);
If &url = "" Then
   throw CreateException(148, 2, "Can not send to OptEngine");
End-If;

/* Specify the Application Server domain URL.
   (This was specified in Step 1 in this example.)
*/
&response = &MSG.SyncRequest(%LocalNode, &url);

If &response.ResponseStatus = 0 Then
   &stName = &response.GetRowset().GetRow(1).GetRecord(Record.PT_OPTPARMS).Get⇒
Field(Field.PARMKEY).Value;
   &stVal = Value(&response.GetRowset().GetRow(1).GetRecord(Record.PT_⇒
OPTPARMS).GetField(Field.VALUE).Value);

   If &stName = "STATUS" And
         &stVal = %OptEng_Fail Then
      throw CreateException(148, 2, "Can not send to OptEngine");
   End-If;

   /* Check Detailed msg here */
```

```
   If &isSync = "Y" And
         &stVal = %OptEng_Success Then

      &respRS = &response.GetRowset();
      &rowNum = &respRS.ActiveRowCount;
      For &iloop = 1 To &rowNum
         &msgRec = &respRS.GetRow(&iloop).GetRecord(Record.PT_OPTDETMSGS);
         If (&msgRec.GetField(Field.MSGSET).Value <> 0) Then
            &msgSet = Value(&msgRec.GetField(Field.MSGSET).Value);
            &msgNum = Value(&msgRec.GetField(Field.MSGNUM).Value);
            &parm1 = &msgRec.GetField(Field.MSGPARM1).Value;
            &parm2 = &msgRec.GetField(Field.MSGPARM2).Value;
            &parm3 = &msgRec.GetField(Field.MSGPARM3).Value;
            &parm4 = &msgRec.GetField(Field.MSGPARM4).Value;
            &parm5 = &msgRec.GetField(Field.MSGPARM5).Value;
            &string = MsgGetText(&msgSet, &msgNum, "Message Not Found", &parm1,⇒
 &parm2, &parm3, &parm4, &parm5);

         End-If;
      End-For;

   End-If;

End-If;
```

Editing the Response PeopleCode

The OPT_CALL message serves as a template. It is delivered to work with the PT_OPTCALL Application Engine program. You can edit the program to suit your needs, or use it as a guide when creating your own response message program.

OPT_CALL Message Program

The OPT_CALL message OnRequest PeopleCode program shows application messages for lights-out mode.

Depending upon the request message, the OnRequest PeopleCode calls appropriate optimization functions and methods to perform these tasks, and sends a response message containing the returned status and detailed messages from the optimization functions and methods.

You can use the OPT_CALL message OnRequest PeopleCode as a template to create your own response message PeopleCode program. For example, you can edit it to run an optimization transaction, which is shown below as an example. This example is edited to match the examples for step 1 and step 3 in the PT_OPTCALL program.

Processing the Transaction Parameters

Edit the OnRequest PeopleCode for application message OPT_CALL to enter the name of your optimization transaction and the name/value pairs for its parameters. In this example, the transaction name is TEST_LONG_TRANS, the first parameter name/value pair is &delayParm and &delay (maps to Delay_in_Secs from the request message), and the second parameter name/value pair is &sleepParm and &isSleep (maps to Sleep0_Work1 from the request message).

The parameter values are stored as strings in step 3 of the Application Engine program. You may need to convert them here to your desired format. Here is a section of the application program showing the places to edit.

```
If &trans = "TEST_LONG_TRANS" Then
   &REC = &rs.GetRow(6).PT_OPTPARMS;
   &delayParm = &REC.PARMKEY.Value;
   &delay = Value(&REC.VALUE.Value);

   &REC = &rs.GetRow(7).PT_OPTPARMS;
   &sleepParm = &REC.PARMKEY.Value;
   &isSleep = Value(&REC.VALUE.Value);

   &myopt = GetOptEngine(&inst, &detStatus);
   If (&myopt = Null) Then
      &optstatus = %OptEng_Fail;
   End-If;

   If &myopt <> Null And &isSync = "Y" Then
      &optstatus = &myopt.RunSynch(&trans, &delayParm, &delay, &sleepParm, &isSleep⇒
);
      &detStatus = &myopt.DetailedStatus;
      End-If;

   If &myopt <> Null And &isSync = "N" Then
      &myopt.ProcessInstance = &procInst;
      &optstatus = &myopt.RunASynch(&trans, &delayParm, &delay, &sleepParm, &is⇒
Sleep);
      &detStatus = &myopt.DetailedStatus;
   End-If; /* iif myopt=null */
End-If;
```

Building a Status Response Message

This section shows the a response message to send a status message for the OPT_CALL message in the application server.

```
/* Insert detailed status and detailed msgs into Response msg rowset */
&respRS = &response.GetRowset();
&respRS.GetRow(1).GetRecord(Record.PT_OPTPARMS).GetField(Field.PARMKEY).Value =⇒
 "STATUS";
&respRS.GetRow(1).GetRecord(Record.PT_OPTPARMS).GetField(Field.VALUE).Value =⇒
 String(&optstatus);

&respRS.InsertRow(1);
&respRS.GetRow(2).GetRecord(Record.PT_OPTPARMS).GetField(Field.PARMKEY).Value =⇒
 "DETAILED_STATUS";
&respRS.GetRow(2).GetRecord(Record.PT_OPTPARMS).GetField(Field.VALUE).Value =⇒
 String(&detStatus);
```

Building a Detailed Response Message

This section shows a response message to send a detailed message for the OPT_CALL message on the application server.

```
/*Either optcmd or inst is not passed in correctly, or optengine is not loaded⇒
/created correctly */
If &myopt = Null Then
   &msgRec = &respRS.GetRow(1).GetRecord(Record.PT_OPTDETMSGS);
   If &isParmBad = True Then
      &msgRec.GetField(Field.MSGSET).Value = 148;
      &msgRec.GetField(Field.MSGNUM).Value = 505;
   End-If;
End-If;

/* If it is sync transaction, insert DetailMsg to response msg */
If &myopt <> Null And
      &isSync = "Y" And
      &optcmd = "RUN" And
      &optstatus = %OptEng_Success Then
   &arrArray = &myopt.DetailMsgs;
   For &iloop = 1 To &arrArray.Len
      /* First two rows have been inserted because of PT_OPTPARMS for two status⇒
 codes */
      If &iloop > 2 Then
         &respRS.InsertRow(&iloop - 1);
      End-If;

      &msgRec = &respRS.GetRow(&iloop).GetRecord(Record.PT_OPTDETMSGS);
      &msgRec.GetField(Field.MSGSET).Value = String(&arrArray [&iloop][1]);
      &msgRec.GetField(Field.MSGNUM).Value = String(&arrArray [&iloop][2]);
      &msgRec.GetField(Field.PARMCOUNT).Value = String(&arrArray [&iloop][3]);
      &msgRec.GetField(Field.MSGPARM1).Value = String(&arrArray [&iloop][4]);
      &msgRec.GetField(Field.MSGPARM2).Value = String(&arrArray [&iloop][5]);
      &msgRec.GetField(Field.MSGPARM3).Value = String(&arrArray [&iloop][6]);
      &msgRec.GetField(Field.MSGPARM4).Value = String(&arrArray [&iloop][7]);
      &msgRec.GetField(Field.MSGPARM5).Value = String(&arrArray [&iloop][8]);
   End-For;

End-If;
```

Optimization Built-in Functions

This section discusses the optimization functions. The functions are discussed in alphabetical order.

CreateOptEngine

Syntax

CreateOptEngine(*probinst*, {%Synch | %ASynch}[, *&detailedstatus*] [, *processinstance*])

Description

The CreateOptEngine function instantiates an OptEngine object, loads an optimization engine with a problem instance and returns a handle to it.

Parameters

Parameter	Description
Probinst	Enter the problem instance ID, which is a unique ID for this problem instance in this optimization engine. This is supplied by users when they request that an optimization be run.
%Synch | %Asynch	Use these system variables to make the optimization engine synchronous or asynchronous.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The problem instance ID passed to the function is invalid. • %OptEng_Exists: An optimization engine instance already exists and is loaded. • %OptEng_Method_Disabled: A method is disabled or not valid. • %OptEng_DB_Updates_Pending: indicates that database updates are pending.
processinstance	Enter the process instance id. You use this parameter only with lights-out processing, most likely with the subscription PeopleCode for application message. **Note.** This optional parameter is positional. If you use it, you must also use the *&detailedstatus* parameter. See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," Using Lights-Out Mode with Optimization. The state record that you use with Application Engine contains the process instance id. See *Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Engine*, "Developing Efficient Programs," Using State Records.

Returns

If successful, CreateOptEngine returns an OptEngine PeopleCode object. If the function fails, it returns a null value. Examine the optional status reference parameter in case of a Null return for additional information regarding the failure.

Example

An OptEngine object variable can be scoped as Local, Component, or Global.

You declare OptEngine objects as type OptEngine. For example:

```
Local OptEngine &MyOptEngine;
Component OptEngine &MyOpt;
Global OptEngine &MyOptEng;
```

The following example loads an optimization engine with the problem instance:

```
Local OptEngine &myopt;
Local string &probinst;
Local string &transaction;
Local integer &detailedstatus;

&probinst = GetRecord(Record.PSOPTPRBINST).GetField(Field.PROBINST).Value;
&myopt = CreateOptEngine(&probinst, %Synch);
```

The following example shows the use of the optional status parameter:

```
&myopt = CreateOptEngine(&probinst, %Synch, &detailedstatus);
if &myopt = Null then
    if &detailedstatus = %OptEng_Invalid_Piid then
        /*perform some action */
    end_if;
end_if;
```

CreateOptInterface

Syntax

CreateOptInterface()

Description

The CreateOptInterface function instantiates an OptInterface object.

Note. You can use this function — and the OptInterface methods — only within an application class that you extend from the OptBase application class, or within PeopleCode that you call from that application class. This ensures that the OptInterface PeopleCode runs only on the optimization engine.

Parameters

None.

Returns

If successful, CreateOptInterface returns an OptInterface PeopleCode object. If the function fails, it returns a null value.

Example

You declare OptInterface objects as type OptInterface. For example:

```
Local OptInterface &MyOptInterface;
Component OptInterface &MyOptInt;
Global OptInterface &MyOptInt;
```

The following example instantiates an OptInterface object:

```
Local OptInterface &myInterface;
```

```
Int &status;

&myInterface = CreateOptInterface(&addtionalStatus);
if (&myInterface != NULL) then
    &status = &myInterface.ActivateModel("RMO_TEST");
        if (&status = %Opt_Fail) then
            /* examine &myInterface.DetailedStatus for reason */
            ...
        end-if;
else
    /* CreateOptInterface has returned NULL */
    /* take some corrective action here */
        ...
end_if;
```

DeleteOptProbInst

Syntax

DeleteOptProbInst(*probinst*[, *&detailedstatus*])

Description

The DeleteOptProbInst function deletes the problem instance ID from PeopleTools metadata. This function can be called only inside FieldChange, PreSave and PostSave PeopleCode events, and in Application Message Subscription PeopleCode and Workflow.

Note. Use this function to delete the problem instance ID after deleting data in optimization application tables for this problem instance.

Parameters

Parameter	Description
probinst	Enter the problem instance ID to delete.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The problem instance ID passed to the function is invalid.

Returns

Returns %OptEng_Success if successful; otherwise returns %OptEng_Fail.

Example

The following example deletes the instance for a problem type:

Note. Whenever you add records to a problem type, you must call DeleteOptProbInst to delete the old problem type instances and then call InsertOptProbInst to recreate them.

```
Local string &probinst;
Local string &probtype;
Local integer &ret;
&probinst = "PATSMITH";
&probtype = "QEOPT";
&ret = DeleteOptProbInst(&probinst, &probtype);
If &ret <> %OptEng_Success  Then
   QEOPT_WRK.MESSAGE_TEXT = "Delete of problem instance " | &probinst | " failed.";
Else
   QEOPT_WRK.MESSAGE_TEXT = "Problem Instance " | &probinst | " deleted.";
End-If;
```

The following example shows the use of the optional status parameter:

```
Local integer &detailedstatus;
&ret = DeleteOptProbInst(&probinst, &probtype, &detailedstatus);
If &ret <> %OptEng_Success  AND &detailedstatus=%OptEng_Invalid_Piid then
   QEOPT_WRK.MESSAGE_TEXT = "Delete of problem instance " | &probinst | " failed
for bad piid.";
Else
   QEOPT_WRK.MESSAGE_TEXT = "Problem Instance " | &probinst | deleted.";
End-If;
```

GetOptEngine

Syntax

GetOptEngine(*probinst*[,*&detailedstatus*])

Description

The GetOptEngine function returns a handle to an optimization engine that is already loaded with the problem instance.

Parameters

Parameter	Description
probinst	Enter the problem instance ID, which is unique ID for this problem instance in this optimization engine.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The problem instance ID passed to the function is invalid.

Returns

Returns an OptEngine PeopleCode object if successful, a null value otherwise.

Example

The following example causes an optimization engine to shut down its problem instance:

```
Global string &probinst;
Local OptEngine &myopt;
Local integer &status;

&myopt = GetOptEngine(&probinst);
If &myopt <> NULL then
&status = &myopt.ShutDown();
QEOPT_WRK.MESSAGE_TEXT = "Problem Instance ID " | &probinst
 | " has been shutdown successfully.";
End-if;
```

Or, you can use the optional status parameter:

```
&myopt = GetOptEngine(&probinst, &detailedstatus);
if &myopt=NULL and &detailedstatus=%OptEng_Invalid_Piid then
     /* perform some action */
End-if;
```

GetOptProbInstList

Syntax

GetOptProbInstList(*ProblemType* , *OutputErrorCode* [, *Prefix*] [, *&detailedstatus*])

Description

The GetOptProbInstList function gets the list of all problem instance IDs in a problem type.

Parameters

Parameter	Description
ProblemType	Enter the name of the problem type that you created in PeopleSoft Application Designer.
OutputErrorCode	Future use. Always returns zero.
Prefix	(Optional) Enter a string. If used, this prefix causes the returned list to include only the problem instance IDs that start with this prefix. If not used, all the problem instance IDs in the problem type are returned.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The problem instance ID passed to the function is invalid.

Returns

Returns an array of strings containing the optimization problem instance list.

Example

The following example shows the usage of GetOptProbInstList to fill the display field on a page:

```
Global string &probinst;
Local integer &detailedstatus;
Local integer &iloop;
Local array of string &instarray;

QEOPT.OPERATOR = %UserId;

&instarray = GetOptProbInstList(QEOPT.PROBTYPE, &detailedstatus, &ret);

If &ret <> %OptEng_Success Then
   QEOPT_WRK.MESSAGE_TEXT = "Could not get problem instances
 for problem type " | QEOPT.PROBTYPE ;
Else
   For &iloop = 1 To &instarray.Len
      QEOPT_WRK.MESSAGE_TEXT = QEOPT_WRK.MESSAGE_TEXT | &instarray[&iloop] | " ";
   End-For;
End-If;
```

The following example shows the use of the optional status parameter:

```
&instarray = GetOptProbInstList(QEOPT.PROBTYPE, &detailedstatus );
If &ret <> %OptEng_Success and &detailedstatus=%OptEng_Invalid_Piid Then
   QEOPT_WRK.MESSAGE_TEXT = "Could not get problem instances for problem type "
| QEOPT.PROBTYPE | "because bad piid" ;
End-If;
```

InsertOptProbInst

Syntax

InsertOptProbInst(*probinst, ProblemType*[, *&detailedstatus*] [,*Description*])

Description

The InsertOptProbInst function inserts a new problem instance ID into the PeopleTools metadata.

The InsertOptProbInst function can be called only inside FieldChange, PreSave and PostSave PeopleCode events, and in Application Message Subscription PeopleCode and Workflow.

Note. You must use this function to create the problem instance ID before inserting data into optimization application tables for this problem instance.

Parameters

Parameter	Description
probinst	Enter the problem instance ID to be inserted into the problem type.
ProblemType	Enter the name of the problem type that you created in PeopleSoft Application Designer.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The problem instance ID passed to the function is invalid.
Description	Specify a description for the problem instance. This parameter takes a string value.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

```
Local string &probinst;
Local string &probtype;
Local integer &ret;
Local integer &detailedstatus;

&probinst = "PATSMITH";
&probtype = "QEOPT";
&probDescr = "New QEOPT instance";
```

```
&ret = InsertOptProbInst(&probinst, &probtype, &probDescr);
If &ret <> %OptEng_Success Then
   QEOPT_WRK.MESSAGE_TEXT = "Insert of problem instance "
 | &probinst | " failed.";
Else
   QEOPT_WRK.MESSAGE_TEXT = "Problem Instance " | &probinst | " created.";
End-If;
```

The following example shows the use of the optional status parameter:

```
&ret = InsertOptProbInst(&probinst, &probtype, &detailedstatus);
If &ret <> %OptEng_Success and &detailedstatus=%OptEng_Invalid_Piid Then
   QEOPT_WRK.MESSAGE_TEXT = "Insert of problem instance "
 | &probinst | " failed for bad piid.";
End-if;
```

IsValidOptProbInst

Syntax

IsValidOptProbInst(*probinst* [, *&status*])

Description

IsValidOptProbInst determines if a given problem instance exists in the optimization metadata.

Parameters

Parameter	Description
probinst	Enter the problem instance ID to be inserted into the problem type.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The problem instance ID passed to the function is invalid.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

```
Local string &probinst;
Local integer &detailedstatus;
```

```
Local integer &ret;

&probinst = "PATSMITH";
&ret = IsValidOptProbInst(&probinst, &detailedstatus);
If &ret <> %OptEng_Success and &detailedstatus=%OptEng_Invalid_Piid Then
   <perform some action>
End-if;
```

OptEngine Class Methods

This section discusses the optimization methods for the OptEngine PeopleCode class. The methods are listed in alphabetical order.

CheckOptEngineStatus

Syntax

CheckOptEngineStatus()

Description

The CheckOptEngineStatus method returns the status of the optimization engine, using a combination of its return value and the DetailedStatus OptEngine class property. Keep the following in mind:

- The value returned by CheckOptEngineStatus is the operational status of the optimization engine.
- The DetailedStatus property indicates the completion status of the OptEngine method call CheckOptEngineStatus.

For example, CheckOptEngineStatus can return %OptEng_Idle and DetailedStatus is %OptEng_Success. For CheckOptEngineStatus, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.

Note. Before this method is called, the CreateOptEngine or GetOptEngine must be called.

Returns

Returns an integer for the status of the optimization engine. These numbers are message IDs belonging to message set 148 in the message catalog.

Numeric Value	Constant Value	Description
21	%OptEng_Not_Loaded	The optimization engine process is running, but is not currently loaded with an application problem.
22	%OptEng_Busy_Loading	The optimization engine is busy loading an application problem. It will not accept transaction requests until loading completes.

Numeric Value	Constant Value	Description
23	%OptEng_Idle	The optimization engine is loaded with an application problem and waiting for a transaction request.
24	%OptEng_Busy	The optimization engine is busy processing a transaction request for the loaded application problem. It will not accept additional transaction requests until the current one completes.
25	%OptEngine_Must_Reboot	A fatal error has occurred and the optimization engine must be shut down and rebooted.
26	%OptEng_Unknown	An error has occurred. The optimization engine status cannot be determined.

Example

This PeopleCode example shows optimization engine status being checked:

```
Local OptEngine &myopt;
Local string &probinst;
Local integer &status;
&myopt = GetOptEngine("PATSMITH");
/* Initialize the DESCRLONG field in the QE_FUNCLIB_OPT record to null.  */
GetLevel0().GetRow(1).GetRecord(Record.QE_FUNCLIB_OPT).DESCRLONG.Value = "";
&status = &myopt.CheckOptEngineStatus();
GetLevel0().GetRow(1).GetRecord(Record.QE_FUNCLIB_OPT).DESCRLONG.Value = "Opt⇒
Engine status = " | MsgGet(148, &status, "Could not send to the OptEngine.");
```

You can also retrieve the detailed status:

```
Local integer &detailedstatus
&status = &myopt.CheckOptEngineStatus();
&detailedstatus = &myopt.DetailedStatus;
```

FillRowset

Syntax

FillRowset(*PARAM_NAME, &Rowset*[, *&functionstatus*])

Description

This method gets the value of a transaction output parameter that is a rowset. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

When using the OptEngine DetailedStatus property, keep the following in mind:

- The value returned by FillRowset is the operational status of the optimization engine.
- The OptEngine DetailedStatus property indicates the completion status of the OptEngine method call FillRowset.

For example, FillRowset returns %OptEng_Fail, and DetailedStatus is %OptEng_Method_Disabled.

For FillRowset, the DetailedStatus property can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled.

  This indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.
&Rowset	Enter the rowset containing the values. This rowset must be a single record rowset, and the record must match the record name associated with the transaction parameter in the problem type definition.
&functionstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • OptEng_Success: The function completed successfully. • OptEng_Fail: The function failed. • OptEng_Method_Disabled: A method is disabled or not valid.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

The following PeopleCode example runs a synchronous optimization transaction named RETURN_MACHINE_UNAVAILABLE. It has these parameters:

- Input: MACHINE_NAME to specify the machine for which we need unavailable times.
- Output: RETURN_TIMES to specify a rowset and MACHINE_WRK record containing the BEGIN_DATE and END_DATE fields.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
```

```
Local Rowset &rs;
&myopt = GetOptEngine("PATSMITH");
&machname = QEOPT_WRK.MACHINE_NAME.Value;
/* Run the RETURN_MACHINE_UNAVAILABLE transaction synchronously with input values.⇒
  */
&status = &myopt.RunSynch("RETURN_MACHINE_UNAVAILABLE", "MACHINE_NAME", &machname);
If Not &status Then
   QEOPT_WRK.MESSAGE_TEXT = " RETURN_MACHINE_UNAVAILABLE transaction failed.";
   Return;
End-If;
/* Get output value from the RETURN_MACHINE_UNAVAILABLE transaction.  */
&rs = CreateRowset(Record.MACHINE_WRK);
&status = &myopt.FillRowset("RETURN_TIMES", &rs);
```

You can also use the [new->] DetailedStatus property as follows:

```
&status = &myopt.FillRowset("RETURN_TIMES", &rs);
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
    /* perform some action */
End-if;
```

GetDate

Syntax

GetDate(*PARAM_NAME*[, *&status*])

Description

This method gets the value of a transaction output parameter with a data type of Date. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The OptEngine DetailedStatus property indicates the completion status of the OptEngine method call GetDate. For GetDate, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

Returns a Date object; use this method when that is the data type of the transaction output parameter value.

Example

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," GetNumber.

GetDateArray

Syntax

GetDateArray(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type Array of Date. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The OptEngine DetailedStatus property indicates the completion status of the OptEngine method call GetDateArray. For GetDateArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

Returns an Array of Date object; use this method when that is the data type of the transaction output parameter value.

Example

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," GetStringArray.

GetDateTime

Syntax

GetDateTime(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type of DateTime. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetDateTime. For GetDateTime, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

Returns a DateTime object; use this method when that is the data type of the transaction output parameter value.

Example

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," GetNumber.

GetDateTimeArray

Syntax

`GetDateTimeArray``(`*`PARAM_NAME`*`)`

Description

This method gets the value of a transaction output parameter with a data type Array of DateTime. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetDateTimeArray. For GetDateTimeArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

Returns an Array of DateTime object; use this method when that is the data type of the transaction output parameter value.

Example

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," GetStringArray.

GetNumber

Syntax

GetNumber(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type of Number. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetNumber. For GetNumber, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

Returns a Number object; use this method when that is the data type of the transaction output parameter value.

Example

The following PeopleCode example runs a synchronous optimization transaction named IS_MACHINE_AVAILABLE. It has these parameters:

- Input MACHINE_NAME to specify the machine.
- Inputs BEGIN_DATE and END_DATE to specify the time slot.
- Output AVAILABLE_FLAG to specify whether the machine is available in that time slot.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local datetime &begindate;
Local datetime &enddate;
&myopt = GetOptEngine("PATSMITH");
&machname = QEOPT_WRK.MACHINE_NAME.Value;
&begindate = QEOPT_WRK.BEGIN_DATE.Value;
&enddate = QEOPT_WRK.END_DATE.Value;
/* Run the IS_MACHINE_AVAILABLE transaction synchronously with input values.  */
&status = &myopt.RunSynch("IS_MACHINE_AVAILABLE", "MACHINE_NAME",
 &machname, "BEGIN_DATE", &begindate, "END_DATE", &enddate);
If Not &status Then
   QEOPT_WRK.MESSAGE_TEXT = "IS_MACHINE_AVAILABLE transaction failed.";
   Return;
End-If;
/* Get output value from the IS_MACHINE_AVAILABLE transaction.  */
QEOPT_WRK.AVAILABLE_FLAG = &myopt.GetNumber("AVAILABLE_FLAG");
```

You can use the DetailedStatus property as follows:

```
QEOPT_WRK.AVAILABLE_FLAG = &myopt.GetNumber("AVAILABLE_FLAG");
if &myopt.DetailedStatus=%OptEng_Fail then
   /* perform some action */
End-if;
```

GetNumberArray

Syntax

GetNumberArray(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type Array of Number. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetNumberArray. For GetNumberArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.

- %OptEng_Method_Disabled

  This indicates that the method is disabled or not valid.

Note. Do not pass an array of type Integer as a transaction parameter. Use an array of type Number instead.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

Returns an Array of Number object; use this method when that is the data type of the transaction output parameter value.

Example

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," GetStringArray.

GetString

Syntax

```
GetString(PARAM_NAME)
```

Description

This method gets the value of a transaction output parameter with a data type of String. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetString. For GetString, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

Returns a String object; use this method when that is the data type of the transaction output parameter value.

Example

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," GetNumber.

GetStringArray

Syntax

`GetStringArray`(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type Array of String. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetStringArray. For GetStringArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

Returns an Array of String object; use this method when that is the data type of the transaction output parameter value.

Example

The following PeopleCode example runs a synchronous optimization transaction named ARE_MACHINES_AVAILABLE. It has these parameters:

- Inputs BEGIN_DATE and END_DATE to specify the time slot.
- Output MACHINE_NAMES to specify the machines available in that time slot.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local array of string &machnames;
Local datetime &begindate;
Local datetime &enddate;
&myopt = GetOptEngine("PATSMITH");
&begindate = QEOPT_WRK.BEGIN_DATE.Value;
&enddate = QEOPT_WRK.END_DATE.Value;
/* Run the ARE_MACHINES_AVAILABLE transaction synchronously with input values.  */
&status = &myopt.RunSynch("ARE_MACHINES_AVAILABLE",
 "BEGIN_DATE", &begindate, "END_DATE", &enddate);
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "ARE_MACHINES_AVAILABLE transaction failed.";
   Return;
End-If;
/* Get output value from the ARE_MACHINES_AVAILABLE transaction.  */
&machnames = &myopt.GetStringArray("MACHINE_NAMES");
```

The following example shows the use of the DetailedStatus property:

```
Local array of string &machnames;
&machnames = &myopt.GetStringArray("MACHINE_NAMES");
if &myopt.DetailedStatus=%OptEng_Fail then
   /* perform some action */
End-if;
```

GetTime

Syntax

GetTime(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type of Time. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetTime. For GetTime, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

Returns a Time object; use this method when that is the data type of the transaction output parameter value.

Example

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," GetNumber.

GetTimeArray

Syntax

`GetTimeArray`(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type Array of Time. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetTimeArray. For GetTimeArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

Returns an Array of Time object; use this method when that is the data type of the transaction output parameter value.

Example

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," GetStringArray.

GetTraceLevel

Syntax

GetTraceLevel(*component*)

Description

GetTraceLevel gets the severity level at which events are logged for a given component.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetTraceLevel. For GetTraceLevel, DetailedStatus can have the value:

- %OptEng_Success.

  This indicates that the function completed successfully.

- %OptEng_Fail.

  This indicates that the function failed.

- %OptEng_Method_Disabled.

  This indicates that the method is disabled or not valid.

- %OptEng_DB_Updates_Pending.

  This indicates that database updates are pending.

Parameters

Parameter	Description
component	Enter one of the following PeopleCode constants: Opt_Engine, Opt_Utility, Opt_Datacache, or Opt_Plugin.

Returns

Returns one of the following.

- %Severity_Fatal
- %Severity_Status
- %Severity_Error
- %Severity_Warn
- %Severity_Info
- %Severity_Trace1
- %Severity_Trace2

Example

```
Local OptEngine &myopt;
Local integer &tracelevel;
```

```
&myopt = GetOptEngine("PATSMITH");

&tracelevel = &myopt.GetTraceLevel(%Opt_Engine);
if &myopt.DetailedStatus = %OptEng_Success then

   if (&tracelevel = %Severity_Info_ then
      winmessage("Severity level for the OptEngine is 'Info'");
   End-if;
End-if;
```

RunAsynch

Syntax

RunAsynch(*TRANSACTION, PARM_PAIRS*)

Description

The RunAsynch method requests the optimization engine to run the transaction in asynchronous mode.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by RunASynch is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call RunASynch.

For example, RunASynch can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For RunASynch, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Parameters

Parameter	Description
TRANSACTION	Enter a string for the name of the transaction to run.
PARAM_PAIRS	Enter the name and value pairs (string name and value) for this transaction. Not used if the transaction has no parameters. Parameters are defined in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

This PeopleCode example runs an asynchronous optimization transaction named SOLVE. It has no input or output parameters. The SOLVE transaction solves the exercise scheduling problem and puts the results into the QE_RWSM_EXERSCH table.

```
Local OptEngine &myopt;
Local integer &status;
&myopt = GetOptEngine("PATSMITH");
/* Run the SOLVE transaction asynchronously with input values.  */
&status = &myopt.RunAsynch("SOLVE");
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "SOLVE transaction failed.";
   Return;
End-If;
```

The following example shows the use of the DetailedStatus property.

```
Local integer &status;
&status = myopt.RunAsynch("SOLVE");
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```

RunSynch

Syntax

RunSynch(*TRANSACTION, PARM_PAIRS*)

Description

The RunSynch method requests the optimization engine to run the transaction in synchronous mode.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by RunSynch is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call RunSynch.

For example, RunSynch can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For RunSynch, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Parameters

Parameter	Description
TRANSACTION	Enter a string for the name of the transaction to run.
PARAM_PAIRS	Enter the name and value pairs (string name and value) for this transaction. Not used if the transaction has no parameters. Parameters are defined in the problem type definition. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Transactions, page 17.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

The following PeopleCode example runs a synchronous optimization transaction named IS_MACHINE_AVAILABLE. It has these parameters:

- Input MACHINE_NAME to specify the machine.
- Inputs BEGIN_DATE and END_DATE to specify the time slot.
- Output AVAILABLE_FLAG to specify whether the machine is available in that time slot.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local datetime &begindate;
Local datetime &enddate;
&myopt = GetOptEngine("PATSMITH");
&machname = QEOPT_WRK.MACHINE_NAME.Value;
&begindate = QEOPT_WRK.BEGIN_DATE.Value;
&enddate = QEOPT_WRK.END_DATE.Value;
/* Run the IS_MACHINE_AVAILABLE transaction synchronously with input values.  */
&status = &myopt.RunSynch("IS_MACHINE_AVAILABLE",
 "MACHINE_NAME", &machname, "BEGIN_DATE", &begindate, "END_DATE", &enddate);
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "IS_MACHINE_AVAILABLE transaction failed.";
   Return;
End-If;
/* Get output value from the IS_MACHINE_AVAILABLE transaction.  */
QEOPT_WRK.AVAILABLE_FLAG = &myopt.GetNumber("AVAILABLE_FLAG");
```

Or, the following example shows the use of the DetailedStatus property.

```
Local integer &status;
&status = myopt.RunSynch("SOLVE");
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```

SetTraceLevel

Syntax

SetTraceLevel(*component*, *severity*)

Description

SetTraceLevel sets the severity level at which events are logged for a given component.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by SetTraceLevel is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call SetTraceLevel.

For example, SetTraceLevel can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For SetTraceLevel, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Parameters

Parameter	Description
component	Use one of the following PeopleCode constants: Opt_Engine, Opt_Utility, Opt_Datacache, or Opt_Plugin.
severity	Use one of the following PeopleCode constants. These options set the degree to which errors are logged. You can set the tracing levels differently for various parts of your program. This enables you to control the amount of trace information that your program generates. The following list shows the order of the severity, starting with the highest level. For example, %Severity_Error logs %Severity_Fatal, %Severity_Status, and %Severity_Error messages, while the system filters out other messages. Keep in mind that the higher the severity, the greater the performance overhead. • %Severity_Fatal • %Severity_Status • %Severity_Error • %Severity_Warn • %Severity_Info • %Severity_Trace1 • %Severity_Trace2

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local datetime &begindate;
Local datetime &enddate;

&myopt = GetOptEngine("PATSMITH");

&status = &myopt.SetTraceLevel(%Opt_Engine, %Severity_Warn);
if &status = %OptEng_Fail then
   <example: notify user that set trace action has failed>
End-if;
```

ShutDown

Syntax

ShutDown()

Description

The ShutDown method requests the optimization engine to shut down.

If the optimization engine cannot be contacted for shutdown, the return status is %OptEng_Fail and the DetailedStatus property is OptEng_Not_Available.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by Shutdown is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call Shutdown.

For example, Shutdown can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For Shutdown, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Note. Before this method is called, CreateOptEngine or GetOptEngine must be called. Call ShutDown to shut down optimization engines even when running in PeopleSoft Application Engine.

Parameters

None.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

This PeopleCode example shows an optimization engine being shut down:

```
Local OptEngine &myopt;
Local integer &status;
&myopt = GetOptEngine("PATSMITH");
/* Shut down the optimization engine */
&status = &myopt.ShutDown();
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "PATSMITH optimization engine shutdown failed.";
```

```
   Return;
Else
   QEOPT_WRK.MESSAGE_TEXT = "PATSMITH optimization engine shutdown successful.";
   Return;
End-If;
```

The following example shows the use of the DetailedStatus property:

```
Local integer &status;
&status = myopt.ShutDown();
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```

OptEngine Class Properties

This section lists the optimization properties for the OptEngine PeopleCode class. The properties are listed in alphabetical order.

DetailMsgs

Description

The DetailMsgs property returns a list of messages generated by an optimization engine. Use DetailMsgs after you use the RunAsynch and RunSynch methods to check the status messages for an optimization transaction.

If the transaction fails, detailed messages are automatically shown to the user. If the transaction succeeds, warnings and informational messages may be generated by the transaction. Use this property to retrieve those messages and make them available to the user.

DetailMsgs provides a two-dimensional array containing the message set ID, the message number in the message catalog, and any arguments. Each row in the two-dimensional array has the following structure:

1. Message set ID.
2. Message number.
3. Number of message arguments.
4. Argument1.
5. Argument2.
6. Argument3.
7. Argument4.
8. Argument5.

A maximum of five arguments is supported for each message.

Note. To hold the property value returned, you need to declare an array of array of type Any.

Note. Before this method is called, you must call CreateOptEngine or GetOptEngine.

Example

```
Local OptEngine &myopt;
Local integer &status;
Local string &piid;

Local string &string;
Local array of array of any &arrArray;

&NEWLINE = Char(10);
&string = "";

&piid = GetRecord(Record.PSOPTPRBINST).GetField(Field.PROBINST).Value;
&myopt = GetOptEngine(&piid);

&status = &myopt.RunSynch("TEST_TRANSACTION");

If (&status = %OptEng_Success) then

&arrArray = &myopt.DetailMsgs;
For &iloop = 1 To &arrArray.Len

   &string = &string | &NEWLINE | MsgGetText(&arrArray [&iloop][1] /*message set*⇒
/,
   &arrArray [&iloop][2] /*message id*/, "Message Not Found",&arrArray[&iloop][4],
   &arrArray [&iloop][5],&arrArray [&iloop][6],
&arrArray [&iloop][7],&arrArray[&iloop][8]);

End-For;

GetLevel0().GetRow(1).GetRecord(Record.QE_FUNCLIB_OPT).DESCRLONG.Value = &string;
End-If;
```

DetailedStatus

Description

The DetailedStatus property contains the detailed execution status of an OptEngine method after the method is executed.

Example

```
Local integer &status;
&status = myopt.ShutDown();
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```

OptBase Application Class

This PeopleCode application class is part of the PT_OPT_BASE application package. It establishes the basic framework for developing PeopleCode that invokes the Optimization PeopleCode plug-in. To use the plug-in, you develop a application class that extends the OptBase application class. OptBase contains the following types of methods:

- A set of base methods that you can extend for the purpose of handling input and output parameters.

  You can use them within any method you develop that corresponds by name to a transaction in a problem type definition. These methods apply to the parameters that are defined for the transaction in the problem type.

- A set of abstract placeholder methods that you can use to implement callback capability.

  You must extend these if you designate one or more records as callback records in your problem type definition, even if you don't add any functionality to the methods.

- An abstract placeholder method, Init, that you can extend if you want to do any preprocessing before your first Optimization PeopleCode plug-in transaction runs.

Note. The problem type definition to which these methods apply is the one that specifies this derived application class.

The CreateOptInterface function is the only optimization built-in function that you can use within an application class that you extend from the OptBase application class, or within PeopleCode that you call from that application class.

Optbase Callback Methods

PeopleSoft Optimization Framework has a built-in callback functionality — when the OptInterface PeopleCode calls an Optimization PeopleCode plug-in transaction, it first determines whether you designated one or more records in your problem type definition as callback records. For each callback record, the framework determines if any the record's database rows have been inserted, deleted, or updated since the optimization datacache was populated. If any changes have occurred, the framework propagates those changes to the datacache before invoking the transaction.

PeopleSoft provides methods that the framework uses to apply its callback functionality. In combination with the framework's callback changes, you might want to perform additional processing for your own purposes, including updating any derived data structures that are used by your optimization application. You can accomplish this by extending the callback methods and adding your own PeopleCode. Each callback method launches under different circumstances.

Note. Don't call any of these methods in your own PeopleCode. They're called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run within each method.

Following is a list of the abstract callback placeholder methods documented as part of the PT_OPT_BASE:OptBase application class:

- OptInsertCallback

  This method launches when the framework propagates to the datacache any database insertions encountered for a callback record.

- OptDeleteCallback

This method launches when the framework propagates to the datacache any database deletions encountered for a callback record.

- OptPreUpdateCallback

  This method launches before the framework propagates each database update encountered for a callback record.

- OptPostUpdateCallback

  This method launches after the framework propagates each database update encountered for a callback record.

- OptRefreshCallback

  This method launches after the framework propagates all database deletions, insertions, and updates encountered for all callback records.

Important! If any record in your problem type definition is designated a callback record, you must ensure that you extend all of the callback methods in your extended class, even if each extended method contains only a Return statement. Otherwise your Optimization PeopleCode plug-in will fail.

See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Records, page 13.

OptBase Class Methods

This section discusses the abstract base class placeholder methods for the PT_OPT_BASE:OptBase application class. The methods are listed in alphabetical order.

GetParmDate

Syntax

```
GetParmDate(parmName, &parmVal)
```

Description

The GetParmDate method retrieves a Date parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmDateArray

Syntax

GetParmDateArray(*parmName, &parmVal*)

Description

The GetParmDateArray method retrieves a Date array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmDateTime

Syntax

GetParmDateTime(*parmName, &parmVal*)

Description

The GetParmDateTime method retrieves a DateTime parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmDateTimeArray

Syntax

GetParmDateTimeArray(*parmName, &parmVal*)

Description

The GetParmDateTimeArray method retrieves a DateTime array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmNumber

Syntax

```
GetParmNumber(parmName, &parmVal)
```

Description

The GetParmNumber method retrieves a Number parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmNumberArray

Syntax

```
GetParmNumberArray(parmName, &parmVal)
```

Description

The GetParmNumberArray method retrieves a Number array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmInt

Syntax

GetParmInt(*parmName, &parmVal*)

Description

The GetParmInt method retrieves an Integer parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify an Integer variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmIntArray

Syntax

GetParmIntArray(*parmName, &parmVal*)

Description

The GetParmIntArray method retrieves a Number array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmString

Syntax

GetParmString(*parmName, &parmVal*)

Description

The GetParmString method retrieves a String parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmStringArray

Syntax

GetParmStringArray(*parmName, &parmVal*)

Description

The GetParmStringArray method retrieves a String array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmTime

Syntax

GetParmTime(*parmName, &parmVal*)

Description

The GetParmTime method retrieves a Time parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmTimeArray

Syntax

GetParmTimeArray(*parmName, &parmVal*)

Description

The GetParmTimeArray method retrieves a Time array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

Init

Syntax

```
Init()
```

Description

The Init method launches when the CreateOptEngine built-in function loads a problem instance that uses the Optimization PeopleCode plug-in.

Use this method to perform additional processing for your own purposes, including checking table data, or any functionality you want to apply before any plug-in transactions run. You accomplish this by adding your own PeopleCode to the extended method.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run before any other code in your extended class.

Note. If you don't extend this method, PeopleSoft Optimization Framework calls its base version from the OptBase application class.

Parameters

None.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptDeleteCallback

Syntax

```
OptDeleteCallback(&Record)
```

Description

The OptDeleteCallback method launches when PeopleSoft Optimization Framework propagates to the datacache any database deletions that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the deletion. You accomplish this by adding your own PeopleCode to the extended method.

Don’t call this method in your own PeopleCode. It’s called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the problem type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	Description
&Record	Specifies a record variable that contains the keys of the data row to be deleted.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptInsertCallback

Syntax

```
OptInsertCallback(&Record)
```

Description

The OptInsertCallback method launches when PeopleSoft Optimization Framework propagates to the datacache any database insertion that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the insertion. You accomplish this by adding your own PeopleCode to the extended method.

Don’t call this method in your own PeopleCode. It’s called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the problem type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	Description
&Record	Specifies a record variable that contains the new data row to be inserted.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptPostUpdateCallback

Syntax

```
OptPostUpdateCallback(&OldRecord, &NewRecord)
```

Description

The OptPostUpdateCallback method launches after PeopleSoft Optimization Framework propagates to the datacache any database update that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might have been affected by the update. You accomplish this by adding your own PeopleCode to the extended method. The parameters provide the previous and current content of the row.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the problem type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	Description
&OldRecord	Specifies a record variable that contains the pre-update content of the data row that was updated.
&NewRecord	Specifies a record variable that contains the post-update content of the data row that was updated.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptPreUpdateCallback

Syntax

```
OptPreUpdateCallback(&OldRecord, &NewRecord)
```

Description

The OptPreUpdateCallback method launches before PeopleSoft Optimization Framework propagates to the datacache any database update that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the update. You accomplish this by adding your own PeopleCode to the extended method. The parameters provide the current and future content of the row.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the problem type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	Description
&OldRecord	Specifies a record variable that contains the pre-update content of the data row to be updated.
&NewRecord	Specifies a record variable that contains the post-update content of the data row to be updated.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptRefreshCallback

Syntax

```
OptRefreshCallback()
```

Description

The OptRefreshCallback method launches after PeopleSoft Optimization Framework propagates to the datacache all database insertions, deletions, and updates that it encounters for all callback records.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the modifications. You accomplish this by adding your own PeopleCode to the extended method.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the problem type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

None.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDate

Syntax

```
SetOutputParmDate(parmName, &parmVal)
```

Description

Use the SetOutputParmDate method to pass a Date parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDateArray

Syntax

```
SetOutputParmDateArray(parmName, &parmVal)
```

Description

Use the SetOutputParmDateArray method to pass a Date array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDateTime

Syntax

```
SetOutputParmDateTime(parmName, &parmVal)
```

Description

Use the SetOutputParmDateTime method to pass a DateTime parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDateTimeArray

Syntax

SetOutputParmDateTimeArray(*parmName, &parmVal*)

Description

Use the SetOutputParmDateTimeArray method to pass a DateTime array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmNumber

Syntax

SetOutputParmNumber(*parmName, &parmVal*)

Description

Use the SetOutputParmNumber method to pass a Number parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmNumberArray

Syntax

```
SetOutputParmNumberArray(parmName, &parmVal)
```

Description

Use the SetOutputParmNumberArray method to pass a Number array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmInt

Syntax

```
SetOutputParmInt(parmName, &parmVal)
```

Description

Use the SetOutputParmInt method to pass an Integer parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify an Integer variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmIntArray

Syntax

SetOutputParmIntArray(*parmName, &parmVal*)

Description

Use the SetOutputParmIntArray method to pass a Number array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmString

Syntax

SetOutputParmString(*parmName, &parmVal*)

Description

Use the SetOutputParmString method to pass a String parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmStringArray

Syntax

```
SetOutputParmStringArray(parmName, &parmVal)
```

Description

Use the SetOutputParmStringArray method to pass a String array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmTime

Syntax

```
SetOutputParmTime(parmName, &parmVal)
```

Description

Use the SetOutputParmTime method to pass a Time parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmTimeArray

Syntax

SetOutputParmTimeArray(*parmName, &parmVal*)

Description

Use the SetOutputParmTimeArray method to pass a Time array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptInterface Class Methods

This section discusses the optimization methods for the OptInterface PeopleCode class. The methods are listed in alphabetical order.

Note. You can use the OptInterface class methods only within an application class that you extend from the OptBase application class, or within PeopleCode that you call from that application class. This ensures that the OptInterface PeopleCode runs only on the optimization engine.

ActivateModel

Syntax

ActivateModel(*ModelID*, *SolverSettingID*)

Description

The ActivateModel method designates the specified model and solver setting as active. The model and the solver are initialized and populated with data from the current problem instance.

Note. This method fails if the specified model (and by extension, one of its solver settings) is already active. If you want to activate a different solver setting for the same model, you must first deactivate the model.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Optimization PeopleCode," DeactivateModel.

Parameters

Parameter	Description
ModelID	Specify the name of the optimization model you want to activate. This must be the name of one of the models associated with the problem type definition.
SolverSettingID	Specify the name of the solver setting you want to activate. This is the name you specified for the solver setting in the problem type definition.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the method fails.

Example

```
Local integer &result;
Local OptInterface &oi = CreateOptInterface();

&result = &oi.ActivateModel("QE_PSA_MODEL", "abc");
```

DeactivateModel

Syntax

DeactivateModel(*ModelID*)

Description

The DeactivateModel method detaches the solver from the specified model.

Parameters

Parameter	Description
ModelID	Specify the name of the optimization model you want to deactivate. This must be the name of one of the models associated with the problem type definition.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the method fails.

Example

```
Local integer &result;
Local OptInterface &oi = CreateOptInterface();

&result = &oi.DeactivateModel("QE_PSA_MODEL");
```

DumpMsgToLog

Syntax

DumpMsgToLog(*LogSeverity*, *Message*)

Description

The DumpMsgToLog method writes the specified status message to the optimization engine log file, with the specified severity.

Parameters

Parameter	Description
LogSeverity	Specify the severity level of the message, as one of the following system constants: • %Severity_Fatal • %Severity_Status • %Severity_Error • %Severity_Warn • %Severity_Info • %Severity_Trace1 • %Severity_Trace2
Message	Specify as a string the text of the log message.

Returns

None.

FindRowNum

Syntax

FindRowNum(*&Record* [, *startrow* [, *endrow* [, *field_list*]]])

Where *field_list* is a list of field names in the form:
[*fieldname1* [, *fieldname2*]]...

Description

The FindRowNum method determines the row number of a row in the datacache rowset. You provide a record with key values, and this method finds the row with the same key values and returns its row number.

Parameters

Parameter	Description
&Record	Specify a record with the same structure as the records that comprise the rowset, with its key fields populated.
startrow	Specify as an integer the starting row number of the search. Specify 0 to search from the first row in the rowset.
endrow	Specify as an integer the ending row number of the search. Specify 0 to search through the last row in the rowset.
fieldname	Specify the name of a field in the input record which contains a value to be matched. You can specify one or more field names, in any order. **Note.** If you use this parameter, the fields specified here are used to search, instead of the record's key fields. Any value that doesn't correspond to a field name is ignored.

Returns

The row number of the row containing the specified key values, or 0 if no row is found.

Example

The following example searches the whole scroll to find the partial key OPT_SITE:

```
Local Record &rec = CreateRecord(Scroll.OPT_TRANSCOST);
Local Optineterface &oi;

&rec.OPT_SITE.value = "New York";
int nRowNum = &oi.FindRowNum(&rec, 0, 0, "OPT_SITE");
```

The following example searches from row 5 to row 15 with the full key values New York and San Jose:

```
Local Record &rec = CreateRecord(Scroll.OPT_TRANSCOST);
Local Optineterface &oi;

&rec.OPT_SITE.value = "New York";
&rec.OPT_STORE.value = "San Jose";
```

```
int nRowNum = &oi.FindRowNum(&rec, 5, 15);
```

GetSolution

Syntax

GetSolution(*ModelID*, *varArrayID*, *skipZero* [, *KeyFieldNames*, *KeyFieldValues* [, *&Solution*]])

Description

The GetSolution method retrieves the model solution values generated by the Solve method.

Parameters

Parameter	Description
ModelID	Specify as a string the name of the optimization model for which you want the solution. This is the name used for the model definition in Application Designer.
varArrayID	Specify as a string the name of the variable array being optimized. Your application documentation should provide this name.
skipZero	Indicate whether solutions with a value of zero should be fetched. This parameter takes a Boolean value: • True: Don't fetch solutions with a zero value. This can increase the performance of the GetSolution method if zero values aren't meaningful. • False: Do fetch solutions with a zero value.
KeyFieldNames and *KeyFieldValues*	Specify a set of key field names as an array of string and a set of key field values as an equal length array of ANY, with one key field value corresponding to each key field name. You use these arrays to restrict the set of returned solutions. Solutions are returned only for model variables with the specified key field values. **Note.** If you provide either of these arrays, you must provide both. You can include each parameter from the variable array at most only once.
&Solution	Specify a rowset to contain the solutions.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the method fails.

Example

```
Local array of string &strArray;
Local array of any &valArray;
Local integer &index;
Local Rowset &rowSet;
```

```
Local integer &result;
Local string &modelId = "QE_PSA_MODEL";
Local string &varArrayName = "X";
Local boolean &bSkipZero = True;

Local OptInterface &oi = CreateOptInterface();

&strArray = CreateArrayRept("", 0);
&valArray = CreateArrayAny();
&rowSet = CreateRowset(Record.QEOPT_VAL_X_WRK);

&strArray [1] = "EMPLID";
&valArray [1] = 1;
&strArray [2] = "ORDER_ID";
&valArray [2] = 23;

/* fetch only the part of the solution where  EMPLID = 1 and ORDER_ID = 23 */
&result = &oi.GetSolution(&modelId, &varArrayName,
 &bSkipZero, &strArray, &valArray, &rowSet);
```

GetSolutionDetail

Syntax

GetSolutionDetail(*ModelID*, *SolutionType*, *Name*, *&Solution*)

Description

The GetSolutionDetail method retrieves the model solution detail of the specified type generated by the Solve method. You can retrieve dual value, slack value, or reduced cost information.

Parameters

Parameter	Description
ModelID	Specify as a string the name of the optimization model for which you want the solution detail. This is the name used for the model definition in Application Designer.
SolutionType	Specify a system constant indicating the type of solution detail you want to retrieve. The value you specify here determines the content of the *Name* and *&Solution* parameters. • %OPT_DUAL: Retrieve the dual value attributes of the specified constraint block. • %OPT_SLACK: Retrieve the slack value attributes of the specified constraint block. • %OPT_RCOST: Retrieve the reduced cost attributes of the specified variable array.
Name	If you specified a *SolutionType* of %OPT_DUAL or %OPT_SLACK, specify here the name of a constraint block from the active model. If you specified a *SolutionType* of %OPT_RCOST, specify here the name of a variable array from the active model.
&Solution	Specify a rowset to contain the solution details. The rowset should have the same key fields as the constraint block or the variable array you specified with the *Name* parameter.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the method fails.

Example

```
Local Rowset &dual_rowset;
Local integer &result;
Local OptInterface &oi = CreateOptInterface();
Local string &modelId = "QE_PSA_MODEL";
Local string &varArrayName = "X";
Local string &constrName = "Constraint_1";

/* fetch dual values for Contraint "Constraint_1"
 in a rowset based on the QEOPT_C1_WRK record */

&dual_rowset = CreateRowset(Record.QEOPT_C1_WRK);
&result = &oi.GetSolutionDetail(&modelId, %Opt_Dual, &constrName, &dual_rowset);
```

IsModelActive

Syntax

IsModelActive(*ModelID*)

Description

Use the IsModelActive method to determine if the model specified by *ModelId* is active before it is used.

Parameters

Parameter	Description
ModelID	Specify the model ID as a string. This is the name used for the model definition in PeopleSoft Application Designer.

Returns

A boolean value: true if the model is active, false otherwise.

RestoreBounds

Syntax

RestoreBounds(*modelID* [,*varArrayID*])

Description

The RestoreBounds method returns the bounding values of the specified variable array or arrays to the current settings in the specified model.

If you previously called the SetVariableBounds method with the *changeModelBounds* parameter set to true for any variable or variable array, those bounding values still apply.

Parameters

Parameter	Description
modelID	Specify as a string the name of the optimization model for which you want to restore the bounding values. This is the name used for the model definition in PeopleSoft Application Designer.
varArrayID	Specify as a string the name of a variable array for which you want to restore the bounding values. Your application documentation should provide this name. If you don't specify a variable array name, the bounding values are restored for all variable arrays in the specified model.

Returns

%OptInter_Success if the method succeeds, %OptInter_Fail otherwise.

SetVariableBounds

Syntax

SetVariableBounds(*modelID*, *varArrayID*, *boundType*, *lowerBound*, *upperBound*, *&keyRecord* [, *changeModelBounds*])

Description

The SetVariableBounds method overrides the bounding values specified for a model variable array, or for a variable within the array.

Parameters

Parameter	Description
modelID	Specify as a string the name of the optimization model for which you want to override the bounding values. This is the name used for the model definition in Application Designer.
varArrayID	Specify as a string the name of the variable array being optimized. Your application documentation should provide this name.
boundType	Specify a system constant indicating which bounding values to override. The value you specify here determines how the *lowerBound* and *upperBound* parameters are applied to the specified model. • %OPT_LOWER_BOUND: Override only the lower bound as specified by the *lowerBound* parameter. The *upperBound* parameter is ignored. • %OPT_UPPER_BOUND: Override only the upper bound as specified by the *upperBound* parameter. The *lowerBound* parameter is ignored. • %OPT_BOUND_BOTH: Override both the lower bound and the upper bound as specified by the *lowerBound* and *upperBound* parameters, respectively.
lowerBound	Specify as a number the lower bound that should be applied to a variable or a variable array if the *boundType* parameter permits the override. You can also set this parameter to one of the following system constants:
upperBound	Specify as a number the upper bound that should be applied to a variable or a variable array if the *boundType* parameter permits the override. You can also set this parameter to one of the following system constants:
&keyRecord	Specify a record with the same key fields as the variable array being optimized. To override the bounding values specified for a single variable within the array, populate the record's key fields to specify the variable. To override the bounding values specified for the entire variable array, set all of the record's fields to a null value. **Note.** You must either provide values for all keys, or set them all to null values.
changeModelBounds	Specify a Boolean value: • true: Indicates that the specified model should be updated in memory to reflect the specified variable bounds. Any problem instance that invokes this model from the active optimization engine is affected by these settings, which are propagated to the solver in memory. This is the default value if you omit this parameter. • false: Indicates that the specified model should not be updated in memory, and that the specified variable bounds apply only to the next time the Solve method is called.

Returns

%OptInter_Success if the method succeeds, %OptInter_Fail otherwise.

Example

```
Local Record &rec;
Local integer &result;
Local OptInterface &oi = CreateOptInterface();
Local float &objval = 0.0;
Local string &modelId = "QE_PSA_MODEL";
```

```
Local string &varArrayName = "X";
Local float &lb = 0.0;
Local float &ub = 0.0;

&rec = CreateRecord(Record.QEOPT_VAL_X_WRK);
&rec.QEOPT_RESINDEX.Value = 1;
&rec.QEOPT_SOLINDEX.Value = 2;
&rec.QEOPT_TIMEINDEX.Value = 3;

&result = &oi.SetVariableBounds(&modelId, &varArrayName,
 %Opt_Upper_Bound, &lb, &ub, &rec, False);
```

SetVariableType

Syntax

SetVariableType(*modelID*, *varArrayID*, *varType*)

Description

Use the SetVariableType method to change the data type of a model variable array.

Parameters

Parameter	Description
ModelID	Specify as a string the name of the optimization model for which you want to change the variable type. This must be the name of one of the models associated with the problem type definition.
varArrayID	Specify as a string the name of the variable array for which you want to change the variable type. Your application documentation should provide this name.
varType	Specify one of the following system constants representing the new variable type: • %Opt_Var_Cont: Represents a continuous variable type, which can be any floating point value. • %Opt_Var_Bin: Represents a binary variable type, for which the value can be only 0 or 1. • %Opt_Var_Int: Represents an integer variable type, which can be any integer.

Returns

%OptInter_Success if the method succeeds, %OptInter_Fail otherwise.

Example

```
Local OptInterface &oi = CreateOptInterface();
Local string &varArrayName = "X";
Local integer &result;

&result = &oi.SetVariableType("QE_PSA_MODEL",  &varArrayName, %Opt_Var_Bin);
```

```
If (&result <> %OptInter_Success) Then
   &oi.DumpMsgToLog(%Severity_Status, "Failed to change variable type ");
End-If;
```

Solve

Syntax

Solve(*modelID*, *SolutionType* [, *&objValue* [, *name-value_pairs*]])

Where *name-value_pairs* is a list of solver setting parameter values in the form:
[*parmname1*, *parmvalue1* [, *parmname2*, *parmvalue2*]]...

Description

The Solve method solves the specified model using the currently active solver settings, and provides an objective value as the solution output. You can override the solver setting parameters. The returned solution status is a predefined system constant.

Parameters

Parameter	Description
ModelID	Specify as a string the name of the optimization model you want to solve. This is the name used for the model definition in Application Designer.
SolutionType	Specify a system constant indicating the type of solution detail you want the model to be solved for. • %OPT_DUAL: Generate dual value attributes. • %OPT_SLACK: Generate slack value attributes. • %OPT_RCOST: Generate reduced cost attributes. You can also combine any or all of these system constants, by connecting them with a plus sign (+), for example: %OPT_DUAL + %OPT_RCOST.
&objValue	Specify a reference to a variable of type float. This variable contains the output objective value produced by the solver upon successfully solving the specified optimization model.
parmname and *parmvalue*	Specify a solver setting parameter ID and value to override the original value you specified for the solver setting in the problem type definition. You can override any or all of the solver setting parameter values. See Chapter 3, "Designing Problem Type Definitions," Configuring Problem Type Models, page 15.

Returns

One of the following system constants:

%Opt_Fail: The solution is not optimal.

%Opt_Optimal: The solution is optimal.

%Opt_Infeasible: The solution is infeasible.

%Opt_Unbounded: The solution is unbounded.

%Opt_Timeup: The solver reached the time limit specified in the solver setting.

%Opt_Iterlimit: The solver reached the limit on the number of iterations specified in the solver setting.

%Opt_LP_Max_Sols: The solver generated maximum number of solutions without improvement.

%Opt_Idle: The solution shows no improvement in a specified time limit.

%Opt_Unknown: The solver status is unknown.

%Opt_MIP_NumSolutions: The specified number of solutions corresponding to an MIP solver reached.

%Opt_MIP_NumNodes: The specified number of nodes corresponding to an MIP solver reached.

%Opt_Aborted: The solver aborted.

%Opt_User_Exit: A user exit was encountered.

%Opt_First_LP_NoOpt: While solving an MIP, the first LP solution obtained was not optimal.

Example

Following is an example of the basic use of the Solve method:

```
Local OptInterface &oi = CreateOptInterface();

Local float &objval = 0.0;
Local integer &result;
Local string &modelId = "QE_PSA_MODEL";
Local string &varArrayName = "X";
Local integer &solType;

&solType = %Opt_RCost + %Opt_Dual + %Opt_Slack;

/* Solve the problem */
&result = &oi.Solve("QE_PSA_MODEL", &solType, &objval);

If & result = %Opt_Optimal Then
   &oi.DumpMsgToLog(%Severity_Warn, " Solution Status =  " Optimal !!!");
Else
   &oi.DumpMsgToLog(%Severity_Warn, " Solution Status =  " | &result );
End-If;
```

Following is an example of a solver setting parameter override:

```
Local OptInterface &oi = CreateOptInterface();
Local float &objval = 0.0;
Local integer &result;

/* This overrides the solver setting for MPS_Filename and generates
 an MPS file called myfile.mps instead of the name specified
 in the current solver setting parameter. */

&result = &oi.Solve("QE_PSA_MODEL", %Opt_Primal, &objval, "MPS_FileName",⇒
 "myfile");
```

CHAPTER 5

Administering Optimization Server Components

This chapter provides an overview of optimization administration and discusses how to:

- Configure an application server domain with optimization engines.
- Configure a PeopleSoft Process Scheduler domain with optimization engines.
- Configure optimization engines.
- Add optimization engines to a domain.

Note. The following sections assume that you have a working knowledge of PeopleSoft application servers and PeopleSoft Process Scheduler.

See Also

Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Process Scheduler

Understanding Optimization Administration

System administrators must perform the procedures in this chapter to configure an application server domain or a PeopleSoft Process Scheduler server that includes optimization engines. You need to enable optimization in a domain and be aware of the configuration options that apply to each engine. In some cases, you might configure the number of optimization engines in a domain to improve performance.

You start and configure optimization engines using the PSADMIN utility.

Configuring an Application Server Domain With Optimization Engines

This section discusses how to

- Enable optimization engines using the Quick Configure menu.
- Enable optimization engines using the PSADMIN interface.
- View optimization engine status.

This section assumes that you already have a working knowledge of PeopleSoft application servers and the PSADMIN utility.

See Also

Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration, "Using the PSADMIN Utility"

Enabling Optimization Engines Using the Quick Configure Menu

To configure the domain for optimization engines using the Quick Configure menu:

1. Run PSADMIN.
2. Complete the following:
    a. Select Application Server from the PeopleSoft Server Administration menu.
    b. Select Create a domain.
    c. Choose a configuration template.
    d. Answer *y* when asked if you would like to configure this domain now.
3. At the Quick-configure menu, enable optimization engines in the Features list, and make any other necessary changes for your domain.
4. Load the configuration.

Enabling Optimization Engines Using the PSADMIN Interface

You can also enable the optimization engines within the standard PSADMIN interface, through the Configure this domain command. After you've specified the appropriate values in the configuration sections, you are asked if you want to configure the optimization engines.

Enter *y* at this prompt.

By enabling fewer server processes, such as publish/subscribe server processes, you reduce the number of server processes that start when the domain boots. This, in turn, makes your configuration simpler while conserving system resources.

If you enter *n* for any of the prompts, the corresponding server process (or set of server processes) is not configured for the domain. If you enter *n* for all of the prompts, your domain contains only the required server processes.

Note. The optimization engines do not interfere with any PeopleSoft functionality.

Viewing Optimization Engine Status

After you have enabled optimization engines within a domain, boot the domain and make sure the optimization server processes start properly.

To view optimization engine status:

1. Boot the application server domain.
    - Select Application Server from the PeopleSoft Server Administration menu.
    - Select Administer a domain, and select the appropriate domain name.
    - Select Boot this domain.
2. After the domain has completed booting, select Domain status menu from the PeopleSoft Domain Administration menu.

3. On the PeopleSoft Domain Status menu, select Server status

   You should see the optimization engine processes (PSOPTENG.exe) in the domain. The number of optimization engine processes that you see depends on the configuration options you've selected.

Note. The application server dynamically scales the number of server processes according to the volume of transaction requests. Optimization engines are a different type of process because they persist in a particular state for long periods. Typically, an application loads an optimization engine with data, solves a mathematical optimization problem, and then enables users to query or modify the solution. Process persistence is essential for solving and interacting with an optimization application. However, process persistence requires a different configuration procedure for scaling optimization engine resources.

Configuring a PeopleSoft Process Scheduler Domain With Optimization Engines

To run optimization engines with PeopleSoft Application Engine through PeopleSoft Process Scheduler, you must configure the Process Scheduler domain for optimization and start the optimization server process, PSAEOSRV. This section discusses how to:

- Enable the optimization server using the Quick Configure menu.
- View optimization server process status.

Enabling the Optimization Server Using the Quick Configure Menu

To configure the domain for optimization engines using the Quick Configure menu:

1. Run PSADMIN.
2. From the PeopleSoft Server Administration menu, select Process Scheduler, then:
   a. Create a domain.
   b. Choose a configuration template.
   c. Answer *y* when asked if you want to configure this domain now.
3. At the Quick-configure menu, enable optimization engines in the Features list.
4. At the Quick-configure menu, make any other appropriate changes and load the configuration.
5. At the Process Scheduler Administration menu, enter *1* to boot the domain.
6. To see the status of PeopleSoft Process Scheduler servers, enter *7*.

Viewing Optimization Server Process Status

To make sure that the required optimization server processes are booting properly, check the server status on the PeopleSoft Process Scheduler server.

To check server status:

1. At the PeopleSoft Process Scheduler Administration menu, select Start a Process Scheduler Server and boot the domain.
2. At the PeopleSoft Process Scheduler Administration menu, select Show status of a Process Scheduler Server.

You should see the following server processes (PSAEOSRV.exe and PSOPTENG.exe) in your server status.

Configuring Optimization Engines

This section provides an overview of platform memory limitations and discusses how to set:

- Domain parameters.
- Trace parameters.
- PSOPTENG parameters.
- Server process options.

Optimization engines can be booted in an application server domain or a PeopleSoft Process Scheduler domain. Configuration options are the same in both domains.

See Also

Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration, "Setting Application Server Domain Parameters"

Understanding Platform Memory Limitations

PeopleTools imposes a memory size limit on optimization engines that depends on the operating system platform where your application server is installed. The current memory limitations are as follows:

- IBM AIX

  By default, optimization engines can consume up to 2 gigabytes (GB) of memory. This is the maximum size available.

- Sun Solaris

  By default, optimization engines can consume up to 4 GB of memory. This is the maximum size available.

- HP Tru64 Unix

  This is a true 64 bit operating system. Optimization engines can theoretically grow to a memory size of 4 GB times 4 GB.

- HP-UX

  By default, optimization engines can consume up to 2 GB of memory. You can increase this limit to 3.8 GB by issuing the following commands:

  1. chatr +q3p enable $PS_HOME/bin/PSOPTENG
  2. chatr +q3p enable $TUXDIR/bin/BBL

- Microsoft Windows

  By default, optimization engines can consume up to 2 GB of memory.

  - You can increase this limit to 3 GB on Microsoft Windows NT Server Enterprise Edition version 4.0 using the 4GT RAM Tuning feature, which is documented in the Microsoft Help and Support knowledge base.

    See http://support.microsoft.com/

- You can increase this limit to 4 GB on Microsoft Windows 2000 Advanced and Datacenter editions, using the Microsoft Windows 2000 Address Windowing Extensions (AWE) API. This is documented in the Microsoft Help and Support knowledge base.

  See http://support.microsoft.com/

• Red Hat Linux

  By default, optimization engines can consume up to 1 GB of memory. This is the maximum size available.

Note. Depending on your hardware and software environment, you may be able to increase the Red Hat Linux memory limit to between 2 GB and 3 GB. This capability is documented on the Red Hat web site.

See http://www.redhat.com/

Setting Domain Parameters

You need to set the Restartable and LogFence parameters.

Restartable

For this parameter, enter *y* or *n*A *y* value indicates that the application server can restart optimization server processes (except the BBL process) if the server dies abnormally.

Note. The value must be *y* so that optimization engines can reboot after a user finishes solving a problem. If you change the parameter to *n* (for example, to change the characteristics of the application servers), then you must override the domain parameter for each optimization engine. Manually edit your UBX file and add the line RESTART=Y after REPLYQ=Y for each optimization engine.

LogFence

Sets an appropriate level of network tracing, ranging from *−100* (suppressing) to *6* (all). The default is *3*, which prints warnings, errors, and status to the log file. The following choices are available for LogFence:

Level	Description
−100	Suppress logging.
−1	Log protocol and memory errors.
0	Log status information.
1	Log general errors.
2	Log warnings.
3	Apply tracing level 1. This is the default level.

Level	Description
4	Apply tracing level 2.
5	Apply tracing level 3.

The trace file is generated in the following directory: *PS_HOME*\appserv*domain*\LOGS\psappsrv.log.

Note. The LogFence parameter establishes a minimum level of tracing in the optimization engines. Additional tracing can be enabled through the optimization trace parameters.

See Also

Chapter 5, "Administering Optimization Server Components," Setting Trace Parameters, page 106

Setting Trace Parameters

You use the Trace section of your domain configuration to specify the TraceOpt option parameter. Trace parameters track the Structured Query Language (SQL) and PeopleCode of your domains.

TraceOpt sets an additional logging level for tracing on all optimization engines. The basic level of tracing is established by the LogFence parameter. Traces are written to the following location: *PS_HOME*/appserv/*domain*/LOGS/optengS*enginenumber*.log.

Trace levels can be set independently for each software component that is running in the optimization engine. The components are OptEngine (lowest level software), DataCache (the optimization engine link to your database), Utilities (general purpose solvers), and Plug-in (the OPI). Each component is given three bits in the 12-bit flag, and they are allocated as follows:

- OptEngine uses the 1st, 2nd, and 3rd bits.
- DataCache uses the 4th, 5th, and 6th bits.
- Utilities use the 7th, 8th, and 9th bits.
- Plug-In uses the 10th, 11th, and 12th bits.

Each set of three bits represents the LogFence for the component using that set, as follows:

Bit position: 3 2 1	LogFence
0 0 0	Severity_Status
0 0 1	Severity_Error
0 1 0	Severity_Warn
0 1 1	Severity_Info
1 0 0	Severity_Trace1

Bit position: 3 2 1	LogFence
1 0 1	Severity_Trace2
1 1 0	Not used
1 1 1	Not used

For example, TraceOpt=3510 sets full trace on all components. Set TraceOpt=0 unless a specific optimization-related problem must be debugged. Full trace logging can produce a large amount of output in the trace file.

Note. The TraceOptMask parameter is not currently used.

Setting PSOPTENG Parameters

The remaining optimization-related parameters are related to the optimization engine processes. Set the parameters in the [PSOPTENG] section of the domain configuration.

Max Instances

Specifies the maximum number of optimization engines allowed in each queue. This parameter does not have any effect at this time, but must be set to at least 1.

Service Timeout

Specifies the number of seconds a PSAPPSRV process waits for an optimization engine to service a request. This parameter is meant to stop long-running processes when the user is waiting for processing to complete. Always make this value less than the service timeout value in the PSAPPSRV section of the domain configuration. If an optimization process needs more time to complete, run the process in an Application Engine program or as an asynchronous optimization transaction.

Opt Max General Services

Specifies the number of distinct SSSQ optimization engines available in the domain. This is used by the application server that supports the PeopleCode OptEngine object. This value needs to match the number of SSSQ queues that have been created in the configuration file.

Setting Server Process Options

After you enter parameter values for the application server, PSADMIN prompts you for server process options, which reduce the number of server processes that start when the domain boots. This makes your configuration simpler while conserving system resources.

If you enter *n* for any service process option prompts, the corresponding server process (or set of server processes) is not configured for the domain. If you enter *n* for all of the prompts, your domain contains only the required server processes.

You can decide to confirm the following actions:

- Configure publish/subscribe servers.
- Move quick PSAPPSRV services into a second server (PSQCKSRV).

- Move long-running SqlQuery services into a second server (PSQRYSRV).
- Configure BEA Jolt.
- Configure JRAD.
- Enable PeopleCode debugging with the PSDBGSRV server.

Adding Optimization Engines to a Domain

The default configuration provides two optimization engines in each domain. You might add more engines if your PeopleSoft installation supports more than two simultaneous optimization activities. For example, two different users might want to solve two unrelated resource scheduling problems using optimization. PeopleSoft Optimization Framework dynamically assigns each user to an optimization engine process. That engine remains dedicated to the user's problem until the user is finished. If a user tries to load a third resource-scheduling problem, PeopleSoft Optimization Framework posts an error message indicating that it's out of available optimization engines.

You can modify the application server template so that you can add more engines.

To modify the application server template:

1. Choose a domain template to modify (large, medium, or small), and then locate the UBX file associated with the template.

   You only modify the number of optimization engines, so base this choice on your needs for application servers and other processes. These instructions assume that you decided to modify the medium domain, which is associated with the UBX file psappsrv.ubx.

2. Open the file $PS_HOME/APPSERV/psappsrv.ubx.
3. Locate the {OPTENG} section where optimization engines are defined.

   Note. There are two OPTENG sections. The correct section lists each engine (PSOPTENG) and seven parameters, such as the server group and server ID.

   For this step, modify the OPTENG section that looks like the following:

```
{OPTENG}
# Optimization Engines, one process per queue.  One PIID in a queue.
# Each queue is explicitly listed with a unique SRVID, RQADDR, and
# command line server name.  Server names are "PSOPTENGn", where 1
# <=n <= 99.
PSOPTENG        SRVGRP=OPTGRP
                SRVID=101
                MIN=1
                MAX=1
                RQADDR="OPTSQ1"
                REPLYQ=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S⇒
 PSOPTENG1"
PSOPTENG        SRVGRP=OPTGRP
                SRVID=102
```

```
                MIN=1
                MAX=1
                RQADDR="OPTSQ2"
                REPLYQ=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S⇒
 PSOPTENG2"
{OPTENG}
```

4. Add a new section for another engine.

   To avoid mistakes, copy and paste the second group of seven lines (from SRVGRP to CLOPT) so that it appears just below the last line of the second PSOPTENG group. Then you have three sections, not two.

   To modify values for the new engine, change the server ID from *102* to *103*, the RQADDR from *OPTSQ2* to *OPTSQ3*, and the command-line server name from *PSOPTENG2* to *PSOPTENG3*.

   The result should look similar to the following:

```
{OPTENG}
# Optimization Engines, one process per queue.  One PIID in a queue.
# Each queue is explicitly listed with a unique SRVID, RQADDR, and command
# line server name.  Server names are "PSOPTENGn", where 1 <=n <= 99.
PSOPTENG        SRVGRP=OPTGRP
                SRVID=101
                MIN=1
                MAX=1
                RQADDR="OPTSQ1"
                REPLYQ=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S⇒
 PSOPTENG1"
PSOPTENG        SRVGRP=OPTGRP
                SRVID=102
                MIN=1
                MAX=1
                RQADDR="OPTSQ2"
                REPLYQ=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S⇒
 PSOPTENG2"
PSOPTENG        SRVGRP=OPTGRP
                SRVID=103
                MIN=1
                MAX=1
                RQADDR="OPTSQ3"
                REPLYQ=Y
                CLOPT="-A -- -C {CFGFILE} -D {$Domain Settings\Domain ID} -S⇒
 PSOPTENG3"
{OPTENG}
```

5. Find the {OPTENG} section where optimization engine services are declared.

   For this step, modify the OPTENG section that looks like the following:

```
{OPTENG}
# Define all optimization services so the SVCTIMEOUT can be set.
```

```
# Use the advertised service names, not services.lst names.
OptLoadOne      SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
OptRunTrans1    SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
OptRunTrans2    SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
OptControlCmd1  SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
OptControlCmd2  SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
{OPTENG}
```

The number of OptRunTrans and OptControlCmd services must be at least as large as the number of optimization engines. Copy and paste to add new services in these two categories, and number them consecutively. The parameters can remain the same for all the services.

```
{OPTENG}
# Define all optimization services so the SVCTIMEOUT can be set.
# Use the advertised service names, not services.lst names.
OptLoadOne      SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
OptRunTrans1    SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
OptRunTrans2    SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
OptRunTrans3    SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
OptControlCmd1  SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
OptControlCmd2  SRVGRP=OPTGRP
```

```
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
OptControlCmd3  SRVGRP=OPTGRP
                LOAD=50 PRIO=50
                SVCTIMEOUT={$PSOPTENG\Service Timeout}
                BUFTYPE="ALL"
{OPTENG}
```

6. Save the file.
7. In the file $PS_HOME/APPSERV/medium.cfx (or small.cfx, or large.cfx), find the Max Instances and Opt Max General Services parameters.
8. Change these lines to reflect the number of engines in your new configuration.

   To change from two to three engines, change both values to *3*.
9. Save the file.
10. Start PSADMIN and create a new domain using the configuration template you modified (in this case the medium template).

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

E

F

G

H

I

P

Q

R

S

T

V

W