

**ORACLE®**

PEOPLESOFT

---

# PeopleTools 8.54: Optimization Framework

---

November 2016

**ORACLE®**

PeopleTools 8.54: Optimization Framework

CDSKU

Copyright © 1988, 2016, Oracle and/or its affiliates. All rights reserved.

## **License Restrictions Warranty/Consequential Damages Disclaimer**

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

## **Warranty Disclaimer**

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

## **Restricted Rights Notice**

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

## **Hazardous Applications Notice**

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

## **Trademark Notice**

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

## **Third Party Content, Products, and Services Disclaimer**

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

## **Documentation Accessibility**

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

## **Access to Oracle Support**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.



# Contents

- Preface.....ix**
  - Understanding the PeopleSoft Online Help and PeopleBooks.....ix
  - PeopleSoft Hosted Documentation.....ix
  - Locally Installed Help.....ix
  - Downloadable PeopleBook PDF Files.....ix
  - Common Help Documentation.....x
  - Field and Control Definitions.....x
  - Typographical Conventions.....x
  - ISO Country and Currency Codes.....xi
  - Region and Industry Identifiers.....xi
  - Using and Managing the PeopleSoft Online Help.....xii
  - PeopleTools Related Links.....xii
  - Contact Us.....xii
  - Follow Us.....xii
- Chapter 1: Getting Started with PeopleSoft Optimization Framework..... 15**
  - PeopleSoft Optimization Framework Overview..... 15
  - PeopleSoft Optimization Framework Implementation..... 15
- Chapter 2: Understanding PeopleSoft Optimization Framework..... 17**
  - Optimization..... 17
  - PeopleSoft Optimization Framework Components..... 17
  - PeopleSoft Optimization Framework System Architecture..... 18
  - Optimization-Based Application Development..... 19
- Chapter 3: Designing Analytic Type Definitions..... 21**
  - Understanding Analytic Type Definitions..... 21
  - Understanding Optimization Application Record Design..... 21
    - Optimization Application Records..... 22
    - Scenario Management..... 22
  - Assigning Permissions for Designing Optimization Records..... 23
  - Creating and Building Optimization Records..... 24
  - Creating Analytic Type Definitions..... 25
    - Defining an Analytic Type..... 25
    - Configuring Analytic Type Records..... 29
    - Configuring Models for Optimization..... 32
    - Associating Analytic Types with Analytic Models..... 35
    - Configuring Analytic Type Transactions..... 36
  - Running the Optimization System Audit..... 38
  - Changing Existing Analytic Type Definitions..... 38
    - Changing Optimization Application Records..... 38
    - Changing Optimization Transactions..... 39
  - Administering Optimization Engines..... 39
    - Setting Up Integration Broker..... 40
  - Updating Solver Licenses..... 40
- Chapter 4: Optimization PeopleCode..... 41**
  - Using Optimization PeopleCode on the Application Server..... 41
  - Using Optimization PeopleCode in an Application Engine Program..... 41
  - Performing Optimization in PeopleCode..... 42
    - Creating New Analytic Instances..... 42

Loading Analytic Instances Into an Analytic Server.....	43
Running Optimization Transactions.....	44
Invoking the Optimization PeopleCode Plug-In.....	45
Shutting Down Optimization Engines.....	45
Deleting Existing Analytic Instances.....	46
Programming for Database Updates.....	46
Using Lights-Out Mode with Optimization.....	46
Understanding Lights-out Mode.....	47
Creating a Request Message.....	48
Creating a Response Message.....	52
Editing the Request PeopleCode.....	53
Editing the Response PeopleCode.....	56
Optimization Built-in Functions.....	58
CreateOptEngine.....	58
CreateOptInterface.....	60
DeleteOptProbInst.....	61
GetOptEngine.....	62
GetOptProbInstList.....	63
InsertOptProbInst.....	64
IsValidOptProbInst.....	66
OptEngine Class Methods.....	67
CheckOptEngineStatus.....	67
FillRowset.....	68
GetDate.....	70
GetDateArray.....	71
GetDateTime.....	71
GetDateTimeArray.....	72
GetNumber.....	73
GetNumberArray.....	74
GetString.....	75
GetStringArray.....	76
GetTime.....	77
GetTimeArray.....	78
GetTraceLevel.....	78
RunAsynch.....	80
RunSynch.....	81
SetTraceLevel.....	82
ShutDown.....	84
OptEngine Class Properties.....	85
DetailMsgs.....	85
DetailedStatus.....	87
OptBase Application Class.....	87
OptBase Class Methods.....	88
GetParmDate.....	89
GetParmDateArray.....	89
GetParmDateTime.....	90
GetParmDateTimeArray.....	90
GetParmNumber.....	91
GetParmNumberArray.....	91
GetParmInt.....	92
GetParmIntArray.....	92

GetParmString.....	93
GetParmStringArray.....	93
GetParmTime.....	94
GetParmTimeArray.....	94
Init.....	95
OptDeleteCallback.....	95
OptInsertCallback.....	96
OptPostUpdateCallback.....	96
OptPreUpdateCallback.....	97
OptRefreshCallback.....	98
SetOutputParmDate.....	98
SetOutputParmDateArray.....	99
SetOutputParmDateTime.....	99
SetOutputParmDateTimeArray.....	100
SetOutputParmNumber.....	100
SetOutputParmNumberArray.....	101
SetOutputParmInt.....	101
SetOutputParmIntArray.....	102
SetOutputParmString.....	102
SetOutputParmStringArray.....	103
SetOutputParmTime.....	103
SetOutputParmTimeArray.....	104
OptInterface Class Methods.....	104
ActivateModel.....	105
ActivateObjective.....	105
DeactivateModel.....	106
DumpMsgToLog.....	107
FindRowNum.....	107
GetSolution.....	108
GetSolutionDetail.....	110
IsModelActive.....	111
RestoreBounds.....	111
SetVariableBounds.....	112
SetVariableType.....	114
Solve.....	115
<b>Chapter 5: Administering Optimization Server Components.....</b>	<b>117</b>
Administering Optimization Server Components.....	117





# Preface

---

## Understanding the PeopleSoft Online Help and PeopleBooks

The PeopleSoft Online Help is a website that enables you to view all help content for PeopleSoft Applications and PeopleTools. The help provides standard navigation and full-text searching, as well as context-sensitive online help for PeopleSoft users.

### PeopleSoft Hosted Documentation

You access the PeopleSoft Online Help on Oracle's PeopleSoft Hosted Documentation website, which enables you to access the full help website and context-sensitive help directly from an Oracle hosted server. The hosted documentation is updated on a regular schedule, ensuring that you have access to the most current documentation. This reduces the need to view separate documentation posts for application maintenance on My Oracle Support, because that documentation is now incorporated into the hosted website content. The Hosted Documentation website is available in English only.

---

**Note:** Only the most current release of hosted documentation is updated regularly. After a new release is posted, previous releases remain available but are no longer updated.

---

### Locally Installed Help

If your organization has firewall restrictions that prevent you from using the Hosted Documentation website, you can install the PeopleSoft Online Help locally. If you install the help locally, you have more control over which documents users can access and you can include links to your organization's custom documentation on help pages.

In addition, if you locally install the PeopleSoft Online Help, you can use any search engine for full-text searching. Your installation documentation includes instructions about how to set up Oracle Secure Enterprise Search for full-text searching.

See *PeopleTools Installation* for your database platform, "Installing PeopleSoft Online Help." If you do not use Secure Enterprise Search, see the documentation for your chosen search engine.

---

**Note:** Before users can access the search engine on a locally installed help website, you must enable the Search portlet and link. Click the Help link on any page in the PeopleSoft Online Help for instructions.

---

### Downloadable PeopleBook PDF Files

You can access downloadable PDF versions of the help content in the traditional PeopleBook format. The content in the PeopleBook PDFs is the same as the content in the PeopleSoft Online Help, but it has a different structure and it does not include the interactive navigation features that are available in the online help.

## Common Help Documentation

Common help documentation contains information that applies to multiple applications. The two main types of common help are:

- Application Fundamentals
- Using PeopleSoft Applications

Most product families provide a set of application fundamentals help topics that discuss essential information about the setup and design of your system. This information applies to many or all applications in the PeopleSoft product family. Whether you are implementing a single application, some combination of applications within the product family, or the entire product family, you should be familiar with the contents of the appropriate application fundamentals help. They provide the starting points for fundamental implementation tasks.

In addition, the *PeopleTools: Applications User's Guide* introduces you to the various elements of the PeopleSoft Pure Internet Architecture. It also explains how to use the navigational hierarchy, components, and pages to perform basic functions as you navigate through the system. While your application or implementation may differ, the topics in this user's guide provide general information about using PeopleSoft Applications.

## Field and Control Definitions

PeopleSoft documentation includes definitions for most fields and controls that appear on application pages. These definitions describe how to use a field or control, where populated values come from, the effects of selecting certain values, and so on. If a field or control is not defined, then it either requires no additional explanation or is documented in a common elements section earlier in the documentation. For example, the Date field rarely requires additional explanation and may not be defined in the documentation for some pages.

## Typographical Conventions

The following table describes the typographical conventions that are used in the online help.

<b><i>Typographical Convention</i></b>	<b><i>Description</i></b>
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.
... (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 (   ).
[ ] (square brackets)	Indicate optional items in PeopleCode syntax.

<b>Typographical Convention</b>	<b>Description</b>
& (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.
=>	This continuation character has been inserted at the end of a line of code that has been wrapped at the page margin. The code should be viewed or entered as a single, continuous line of code without the continuation character.

## ISO Country and Currency Codes

PeopleSoft Online Help topics use International Organization for Standardization (ISO) country and currency codes to identify country-specific information and monetary amounts.

ISO country codes may appear as country identifiers, and ISO currency codes may appear as currency identifiers in your PeopleSoft documentation. Reference to an ISO country code in your documentation does not imply that your application includes every ISO country code. The following example is a country-specific heading: "(FRA) Hiring an Employee."

The PeopleSoft Currency Code table (CURRENCY\_CD\_TBL) contains sample currency code data. The Currency Code table is based on ISO Standard 4217, "Codes for the representation of currencies," and also relies on ISO country codes in the Country table (COUNTRY\_TBL). The navigation to the pages where you maintain currency code and country information depends on which PeopleSoft applications you are using. To access the pages for maintaining the Currency Code and Country tables, consult the online help for your applications for more information.

## Region and Industry Identifiers

Information that applies only to a specific 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 region-specific heading: "(Latin America) Setting Up Depreciation"

### Region Identifiers

Regions are identified by the region name. The following region identifiers may appear in the PeopleSoft Online Help:

- 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 the PeopleSoft Online Help:

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

---

## Using and Managing the PeopleSoft Online Help

Click the Help link in the universal navigation header of any page in the PeopleSoft Online Help to see information on the following topics:

- What's new in the PeopleSoft Online Help.
- PeopleSoft Online Help accessibility.
- Accessing, navigating, and searching the PeopleSoft Online Help.
- Managing a locally installed PeopleSoft Online Help website.

---

## PeopleTools Related Links

[Oracle's PeopleSoft PeopleTools 8.54 Documentation Home Page \(Doc ID 1664613.1\)](#)

"PeopleTools Product/Feature PeopleBook Index" (PeopleTools 8.54: Getting Started with PeopleTools)

[PeopleSoft Information Portal](#)

[My Oracle Support](#)

[PeopleSoft Training from Oracle University](#)

[PeopleSoft Video Feature Overviews on YouTube](#)

---

## Contact Us

[Send us your suggestions](#) Please include release numbers for the PeopleTools and applications that you are using.

---

## Follow Us



Get the latest PeopleSoft updates on [Facebook](#).



Follow PeopleSoft on [Twitter@PeopleSoft\\_Info](https://twitter.com/PeopleSoft_Info).



# Getting Started with PeopleSoft Optimization Framework

---

## PeopleSoft Optimization Framework Overview

PeopleSoft Optimization Framework provides a foundation for building applications that use the optimization-based, decision-making capability in the PeopleTools environment.

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

- Understanding PeopleSoft Optimization discusses optimization and the framework components and architecture, as well as doing optimization-based development.
- Designing Analytic Type Definitions provides overviews of analytic 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.

### Related Links

[Creating Analytic Type Definitions](#)

[PeopleSoft Optimization Framework Components](#)

[PeopleSoft Optimization Framework System Architecture](#)

---

## 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 product documentation for *PeopleTools Installation* for your database platform.

- Establish a user profile that provides access to PeopleSoft Application Designer and any other processes you will use.
- Follow the general overview and instructions in this document to design your application to take advantage of PeopleSoft Optimization Framework, populate the appropriate records, build the application pages, retrieve the result data, as well as configure the application server, the analytic server, and the optimization engines.

**Related Links**

[PeopleSoft Optimization Framework System Architecture](#)



# Understanding PeopleSoft Optimization Framework

---

## 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 meet 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 type of PeopleSoft application server, called an Analytic 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 analytic 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 are able to adapt the plug-in to a variety of optimization tasks.

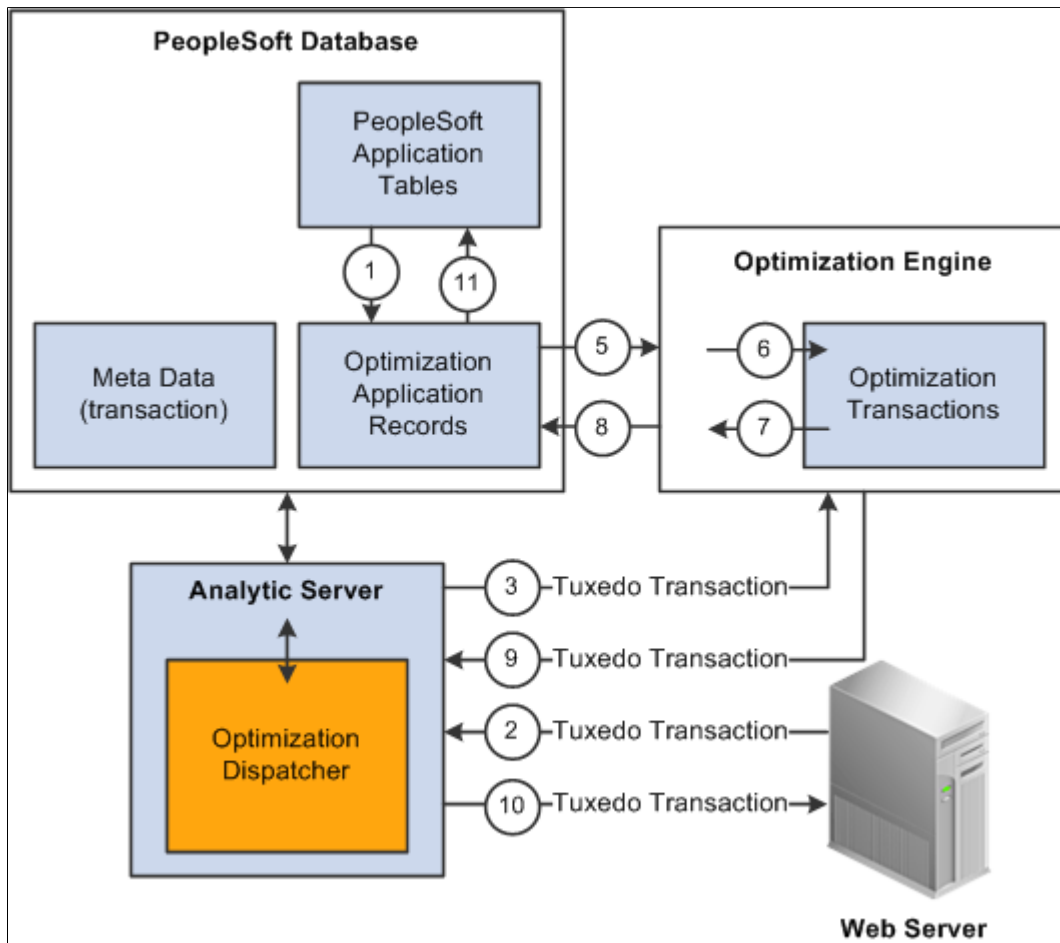
**Note:** An OPI is created by PeopleTools development with support from PeopleSoft application development. An OPI is provided with the installed PeopleSoft applications that use PeopleSoft Optimization Framework. No OPI is in the PeopleTools installation. Your PeopleSoft application documentation discusses 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:

**Image: PeopleSoft Optimization Framework architecture**

The following diagram represents the high level architecture of PeopleSoft Optimization Framework.



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 Oracle Tuxedo to have the analytic server perform a PeopleSoft transaction using optimization.
3. Upon receiving the request, the optimization dispatcher, within the analytic server, locates the correct optimization engine and sends the optimization transaction to it through Oracle Tuxedo.
4. The optimization engine gets the metadata (optimization transaction name, parameters, and data types of the parameters) from the analytic 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 Oracle Tuxedo.
11. After 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 analytic type definition.

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

- a. Create record definitions for the optimization application records and build them to create the database tables.
- b. Create an analytic type definition, including the record definitions that you created and the specifications for the optimization transactions.
- c. If needed, insert one or more *optimization models* into the analytic 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.

---

**Note:** Though you can populate the source data using PeopleSoft Integration Broker, you cannot actually access the analytic server or use analytic or optimization PeopleCode in a messaging PeopleCode program.

---

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 or AnalyticInstance 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 analytic 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.

---

# Designing Analytic Type Definitions

---

## Understanding Analytic Type Definitions

An analytic type definition groups the optimization application records, the optimization transactions, and the Optimization plug-in (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 analytic 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 analytic 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 analytic 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 an analytic instance. You can break down large optimization problems into smaller, more manageable problems (or analytic instances) that can each be solved independently. Individual analytic instances can share common data.

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

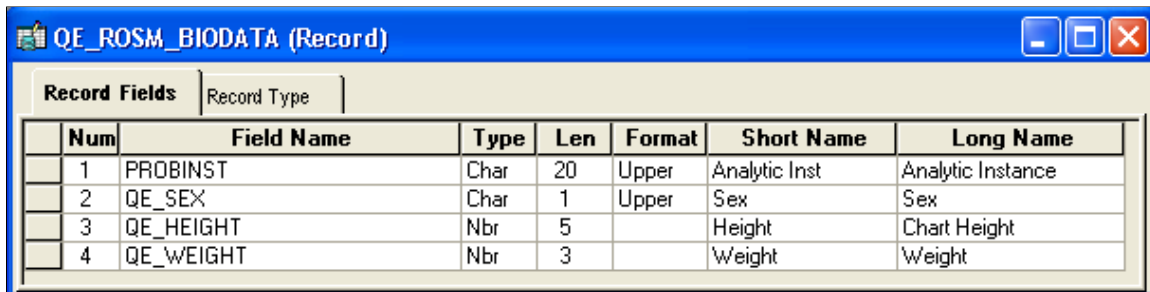
In terms of the exercise example, any number of people might want exercise schedules using the optimization application. Exercise goal data and the optimization-generated exercise schedule data are unique to each person. However, different people share the same set of exercise machines. In this case, it is logical to treat the generation of an individual person's exercise schedule as a separate analytic 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 nonscenario-managed. All scenario-managed records must include the PROBINST field as part of the primary key. This 20-character field identifies data that is specific to an analytic 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 analytic 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 is scenario-managed. Notice the use of the PROBINST field:

**Image: QE\_ROSM\_BIODATA record**

The following diagram illustrates the Record fields in QE\_ROSM\_BIODATA Record.



Num	Field Name	Type	Len	Format	Short Name	Long Name
1	PROBINST	Char	20	Upper	Analytic Inst	Analytic Instance
2	QE_SEX	Char	1	Upper	Sex	Sex
3	QE_HEIGHT	Nbr	5		Height	Chart Height
4	QE_WEIGHT	Nbr	3		Weight	Weight

## Assigning Permissions for Designing Optimization Records

For users to create and build optimization records, they must have access to the Optimization Model Designer in PeopleSoft Application Designer. This is accomplished by providing permission list access to the Optimization Model object.

To assign permissions for designing optimization records:

1. Select PeopleTools, Security, Permission Lists.

The Permission Lists–General page appears.

2. Click the PeopleTools tab.

The Permission Lists–PeopleTools page appears.

3. In the PeopleTools Permissions section under the Application Designer Access check box, click the Definition Permissions link.

The Definition Permissions page appears.

4. In the Object list, locate the Optimization Model object.

5. From the drop-down list select an option:

- *Full Access.* Users can read, create and modify optimization records.
- *No Access.* (Default.) Users cannot view, create, or modify optimization records.
- *Read-only Access.* Users can view optimization records.

6. Click the OK button.

The Definition Permissions page appears.

7. Click the Save button.

---

## Creating and Building Optimization Records

To create and build optimization application records:

1. Create the optimization application record definitions using PeopleSoft Application Designer.

- a. Select Start, Programs, PeopleSoft 8.xx, Application Designer.
- b. Enter your signon information, and click the OK button.

The Application Designer window appears.

- c. Select File, New from the tool menu.
- d. Select the *Record* option, and click the OK button.

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.

Alternatively, use a sub-record definition that is shared by the optimization application record and the delete record.

---

**Note:** Oracle strongly recommends that you keep the optimization application record and its associated optimization delete record in sync with each other.

---

3. For every optimization application record that is readable, associate that record with its optimization delete record using the these steps:

- a. In PeopleSoft Application Designer, 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 can be used by several analytic types. When a record is deleted from an analytic type, the associated delete record is not needed if this record is not used elsewhere.

---

---

## Creating Analytic Type Definitions

This section discusses how to:

- Define an analytic type.
- Configure analytic type records.
- Configure models for optimization.
- Associate Analytic Types with Analytic Models.

---

**Note:** When working with analytic 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 analytic type record list, which is maintained on the Record tab of the analytic type definition.

---

## Defining an Analytic Type

In PeopleSoft Application Designer, select File, New, Analytic Type. A new analytic 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.

This is an example of the analytic type definition:

**Image: Analytic Type – Transactions tab**

This example illustrates the fields and controls on the Analytic Type – Transactions tab.

	Transaction Name	Lock Flag	Description
1	DATA_CACHE_INSPECT	<input type="checkbox"/>	Transaction to inspect a cer
2	DATA_CACHE_INSP_MUL	<input type="checkbox"/>	Transaction to inspect multirows
3	DATA_CACHE_MULT_TES	<input type="checkbox"/>	Insert/update/delete multi rows
4	DATA_CACHE_TEST	<input type="checkbox"/>	Data Cache Test - Transaction which test the insert/delete/update row in d
5	GET_SUMMARY	<input type="checkbox"/>	Get Exercise Summary
6	LICENSE_TEST	<input type="checkbox"/>	Test License code for different math solvers - CPLEX, DASH, etc.
7	PLUGINMGR_TEST_CALL	<input type="checkbox"/>	Test callback
8	PLUGINMGR_TEST_CREA	<input type="checkbox"/>	Create for pluginmgrtest
9	PLUGINMGR_TEST_MISC	<input type="checkbox"/>	Update, insert, etc to cache
10	PLUGINMGR_TEST_MOM	<input type="checkbox"/>	Various actions to datacache
11	PLUGINMGR_TEST_TRAN	<input type="checkbox"/>	Run pluginmgrtest trans
12	PROG_METER_TEST	<input type="checkbox"/>	Tests the progress meter
13	SOLVE	<input type="checkbox"/>	Solve
14	TEST_BAD_OUTPUT	<input type="checkbox"/>	The QEOPT test plug-in will deliberately return an unknown parameter.
15	TEST_DETAILED_MSGS	<input type="checkbox"/>	The QEOPT plug-in will send a range of detailed messages to the AppServer
16	TEST_DUMMY	<input type="checkbox"/>	A dummy transaction referenced by the Problem Type definition but is not ref
17	TEST_EXCEPTION	<input type="checkbox"/>	The QEOPT plug-in throws an exception that is handled by OptEngine logic.
18	TEST_LONG_TRANS	<input type="checkbox"/>	The transaction pretends to work for a specified period of time. Useful for te
19	TEST_OPT_INTERFACE	<input type="checkbox"/>	Test OptInterface
20	TEST_PARAMETERS	<input type="checkbox"/>	Smoke test for parameter passing.
21	TEST_PCODE_EVAL	<input type="checkbox"/>	Testing PeopleCode Eval feature
22	TEST_RETRY_LOAD	<input type="checkbox"/>	
23	TEST_REVERT_TRANS	<input type="checkbox"/>	Test failing a transaction works

To complete the analytic type definition, you should configure the analytic type properties, then insert and configure the records, the optimization models, and the transactions, in that order.

To access the Analytic Type Properties – Attributes dialog:

1. From the Analytic Type – Transaction tab, select File, Definition Properties.

2. Select the Attributes tab.

### Image: Analytic Type Properties – Attributes tab

This example illustrates the fields and controls on the Analytic Type Properties – Attributes tab. You can find definitions for the fields and controls later on this page.

#### PeopleCode Plugin

Select to indicate that the analytic type should use the Optimization PeopleCode plug-in.

Select this check box only if the analytic type is to be used with optimization. If the analytic type is to be used with the analytic calculation engine, do not select this check box.

*Psopidplugin* is automatically entered in the Plugin Library 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 was developed to adapt the Optimization PeopleCode plug-in to your optimization application.

See "Invoking the Optimization PeopleCode Plug-In" (PeopleTools 8.54: PeopleCode API Reference).

<b>Plugin Library Name</b>	<p>Enter the name of the OPI library.</p> <p>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, in Microsoft Windows, the library is libqopt.dll. You would enter only <i>qopt</i> here.</p> <p>If you selected the PeopleCode Plugin check box, this field contains the value <i>psopidplugin</i>, and is read-only.</p>
<b>Plugin Library Version</b>	<p>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.</p>
<b>Message Set ID</b>	<p>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.</p>
<b>Plugin Application Class – Package</b>	<p>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.</p>
<b>Plugin Application Class – Class</b>	<p>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.</p> <p>This class must be a subclass of the PT_OPT_BASE:OptBase application class.</p>
<b>Analytic Instance Application Class – Package</b>	<p>If this analytic type is to be used with the PeopleSoft Analytic Calculation Engine, specify the application package name to associate with this analytic type, that contains the functionality to be used with the analytic type when it is created, deleted, or copied.</p> <p>See "Creating, Deleting, and Copying Analytic Instances" (PeopleTools 8.54: Analytic Calculation Engine).</p>
<b>Analytic Instance Application Class – Class</b>	<p>If this analytic type is to be used with the PeopleSoft Analytic Calculation Engine, specify the name of the class in the application package that contains the Create, Copy, and Delete classes.</p>

## Related Links

[Understanding Analytic Type Definitions](#)

## Configuring Analytic Type Records

To configure analytic type records, in the analytic type definition, select the Record tab, and then select Insert, Record.

### Image: Analytic Type Record Property dialog box

This example illustrates the fields and controls on the Analytic Type Record Property dialog box. You can find definitions for the fields and controls later on this page.

**Analytic Type Record Property**

Record Name: QEOPT\_ASSIGNMNT

Synchronization Order: 9

Description:

Read Once:

Readable:

Writable:

Scenario Managed:

CallBack:

**Record Fields**

	Field Name	Select
1	PROBINST	<input checked="" type="checkbox"/>
2	QEOPT_RR_ID	<input checked="" type="checkbox"/>
3	EMPLID	<input checked="" type="checkbox"/>
4	START_DT	<input checked="" type="checkbox"/>
5	END_DT	<input type="checkbox"/>
6	QEOPT_SCHED_HRS	<input type="checkbox"/>
7	VERSION	<input checked="" type="checkbox"/>

OK Cancel

**Note:** You can access the properties of an existing analytic type record by right-clicking the record and selecting the *Analytic Type Record Properties* option.

**Record Name** Select the record to use in the analytic type definition.

---

**Note:** If you select a derived/work record, remember that its scope in optimization PeopleCode is different from that in other PeopleCode. When you use the CreateOptEngine or CreateAnalyticInstance function, each derived/work record is instantiated at level zero of the analytic 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 analytic instance into the optimization engine.

You cannot select the Writeable check box if the Read Once check box is selected.

The optimization engine reads these records only once during the initial data load. The assumption is 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 needs to be read only 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 topic.

---

**Note:** Oracle recommends that you keep the analytic type records in sync with the optimization delete records.

---

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 contain 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. Records more likely to be readable and writable than 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 analytic instances.

---

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

---

See [Scenario Management](#).

### Callback

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

Your analytic 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 derived data structures of the optimization. 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 "OptBase Application Class" (PeopleTools 8.54: PeopleCode API Reference).

### 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.

When the analytic type definition is saved, if there are fields that have not been selected but are being mapped to a cube or

dimension, an error message is displayed, and you must go back and correct the error before you can save the analytic type definition. If there is a record in the analytic type definition that has none of its fields mapped to any cube or dimension, a warning message is displayed when you try to save the analytic type definition. You can continue to save the analytic type definition after you have acknowledged the warning message; you do not have to change anything in the definition.

## Configuring Models for Optimization

You need to specify and configure analytic type models for optimization only if both of the following conditions are true:

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

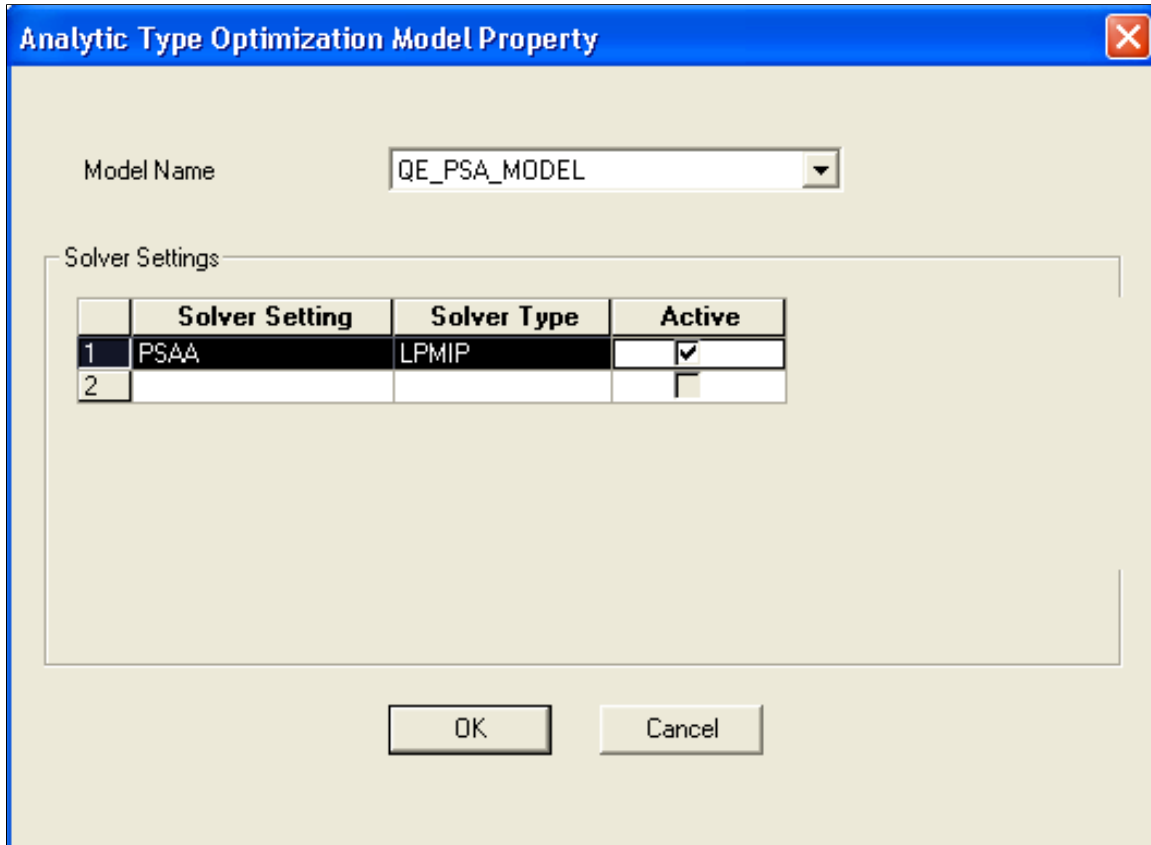
In the analytic type definition, select the Models tab, and then select *Insert, Optimization Model*.



The Analytic Type Optimization Model Property dialog box appears.

**Image: Analytic Type Optimization Model Property dialog box**

This example illustrates the fields and controls on the Analytic Type Optimization Model Property dialog box. You can find definitions for the fields and controls later on this page.



**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 analytic type model by right-clicking the model and selecting the *Analytic Type Model Properties* option.

**Model Name**

Select the optimization model required to implement an optimization application with this analytic 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.

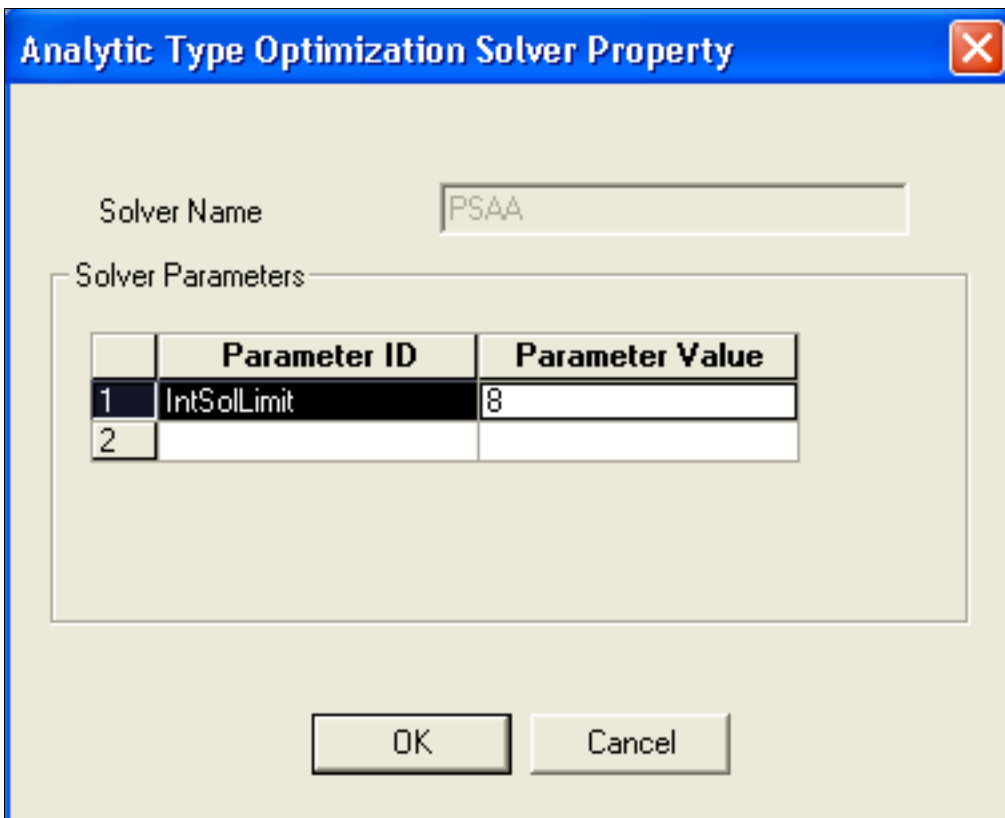
### Configuring Solver Parameters

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

In the Analytic Type Optimization Model Properties dialog box, double-click a solver setting to access the Analytic Type Optimization Solver Property dialog box. This dialog box has a grid with two columns: Parameter ID and Parameter Value:

**Image: Analytic Type Optimization Solver Property dialog box**

This example illustrates the fields and controls on the Analytic Type Optimization Solver Property dialog box.



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 drop-down list box, 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.

### Creating Mathematical Formulation Files

In addition to the analytic server log files, you can also create a mathematical formulation file for debugging. This file is written in either MPS or LP format and can be requested for technical debugging purposes. The file type is generally LP; however, if the system cannot create an LP file it creates an

MPS file. The filename is either *AnalyticType\_AnalyticInst.LP* or *AnalyticType\_AnalyticInst.MPS*, with *AnalyticType* being the name of the analytic type and *AnalyticInst* being the name of the analytic instance ID. This file is generally written to the same directory as the application server log. Also, this directory can be configured in the application server configuration file.

You indicate whether to write this file by specifying a solver parameter.

In the Analytic Type Optimization Model Properties dialog box, double-click a solver setting to access the Analytic Type Optimization Solver Property dialog box. This dialog box has a grid with two columns: Parameter ID and Parameter Value.

Select the *WriteMPS* option for Parameter ID. In the Parameter Value column, enter *1* to write the file or *0* to not write the file.

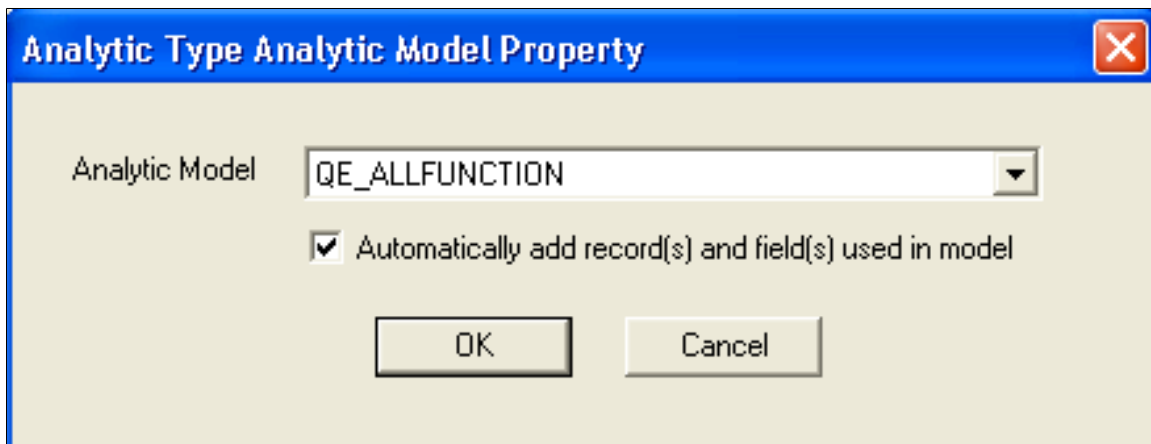
## Associating Analytic Types with Analytic Models

For PeopleSoft Analytic Calculation Engine, you only need to associate an analytic type with an analytic mode.

In the analytic type definition, select the Models tab, and then select *Insert, Analytic Model*.

### Image: Analytic Type Analytic Model Property dialog box

This screenshot shows the fields and controls on the Analytic Type Analytic Model Property dialog box.



Select the name of the analytic model that you want to associate with the analytic type. If you specify to add all the records and fields that are used in the model, they are automatically added to the records on the Records tab.

## Configuring Analytic Type Transactions

In the analytic type definition, select the Transactions tab, and then select **Insert, Transaction**.

### Image: Analytic Type Transaction Property dialog box

This example illustrates the fields and controls on the Analytic Type Transaction Property dialog box. You can find definitions for the fields and controls later on this page.

**Analytic Type Transaction Property**

Transaction Name:   Lock Flag

Description:

Parameter Attributes

	Name	Type	Input/Output	Attributes	Value
1	MACHINE_NAME	String	Input	Required	
2	BEGIN_DATE	DateTime	Input	Required	
3	END_DATE	DateTime	Input	Required	
4	AVAILABLE	Integer	Output	N/A	
5					

OK Cancel

**Note:** You can access the properties of an existing analytic type transaction by right-clicking the transaction and selecting the *Analytic Type Transaction Properties* option.

### Transaction Name

Enter the case-sensitive name of the transaction.

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

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

The transaction name that you specify must be distinct within an analytic 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 "OptBase Application Class" (PeopleTools 8.54: PeopleCode API Reference).

### Lock Flag

Select this option 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 that you specified for this analytic type.

The transaction parameter name can contain up to 20 characters, and it must be distinct within an analytic 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 that you specified for this analytic 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 analytic type definition, run SYSAUDIT with the optimization options selected. This ensures that the definition is valid and consistent.

To run the optimization system audit in the PeopleSoft application:

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

This file contains the audit report for your optimization.

### Related Links

"Running SYSAUDIT" (PeopleTools 8.54: Data Management)

---

## Changing Existing Analytic Type Definitions

This section discusses how to change:

- Optimization application records.
- Optimization transactions.

## Changing Optimization Application Records

To change optimization application records in an analytic type definition:

1. Shut down all the running optimization engines that use this analytic type definition.
2. Shut down other optimization engines if record definitions are being shared by other analytic type definitions.
3. Delete all existing analytic instances using the DeleteOptProbInst PeopleCode function.  
See "DeleteOptProbInst" (PeopleTools 8.54: PeopleCode API Reference).
4. Empty the optimization application tables.
5. Make record definition changes and build the records in PeopleSoft Application Designer.  
See [Creating and Building Optimization Records](#).
6. Open the analytic type in PeopleSoft Application Designer, insert any new records or make appropriate changes to reflect changed record definitions, and save the analytic 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 re-create analytic instances.  
See "InsertOptProbInst" (PeopleTools 8.54: PeopleCode API Reference).

## Changing Optimization Transactions

To change optimization transactions in an analytic type definition:

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

An optimization engine is an instance of an analytic server.

You can use the Analytic Server Administration – Analytic Domain Summary page to administer all optimization engines. To access the Analytic Server Administration – Analytic Domain Summary page from PIA, select PeopleTools, Utilities, Administration, Analytic Server Administration.

See [PeopleSoft Optimization Framework System Architecture](#)

## 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.

The only PeopleSoft Integration Broker elements that are 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 are both active on the Transactions page of the default local node definition.

See "Understanding PeopleSoft Integration Broker" (PeopleTools 8.54: Integration Broker)

---

## 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

---

## 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 calls 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 calls 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 analytic instances.
- Load analytic instances into an analytic server.
- Run optimization transactions.
- Invoke the Optimization PeopleCode plug-in.
- Shut down optimization engines.
- Delete existing analytic instances.
- Program for database updates.

---

**Important!** The optimization PeopleCode classes are not supported on IBM z/OS and Linux for IBM System z platforms.

---

## Creating New Analytic Instances

To create a new analytic instance for an analytic type:

1. Call the function InsertOptProbInst with the analytic type and analytic instance as parameters to create an analytic instance ID.
2. Use Application Engine or a similar mechanism to load the optimization application tables with data.

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

The analytic instance is now ready to be loaded into an analytic server.

---

**Note:** You can load multiple copies of the same analytic instance into multiple instances of an analytic server, provided that each instance of the analytic server resides in a different application server domain. Each analytic instance loaded into a given domain must be unique. Within a given domain, you cannot have the same analytic instance in more than one analytic server. The analytic server 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 analytic instance uses the same database in different domains.

---

## Related Links

[PeopleSoft Optimization Framework System Architecture](#)

[InsertOptProblnst](#)

## Loading Analytic Instances Into an Analytic Server

Use the `CreateOptEngine` function to load an analytic server with an analytic instance. It takes analytic 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 Application Engine.

### Loading Analytic 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 analytic server without blocking the PeopleCode from running (asynchronous mode) on the application server, use the `%Asynch` value for the mode parameter. The analytic server 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 analytic engine is done with the load and whether it was successful. Here is a code example:

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

### Loading Analytic Instances by Running PeopleCode in Application Engine

Both synchronous (`%Synch`) and asynchronous (`%Asynch`) modes block the PeopleCode from running on 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.

## Related Links

[CheckOptEngineStatus](#)

## 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 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 an analytic instance ID.

Use the `CreateOptEngine` function to create the `OptEngine` object for a new optimization engine if the analytic 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 analytic type in 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 Application Engine

Both synchronous (RunSynch) and asynchronous (RunAsynch) methods block the PeopleCode from running on 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.

### Related Links

[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 an analytic type that specifies the Optimization PeopleCode plug-in, by selecting the PeopleCode Plugin check box in the analytic type properties.
- In the analytic type properties, specify the application package and application class that you developed.
- Define transactions in your analytic type definition that correspond to the methods you developed in your application class, with corresponding parameters.

### Related Links

[Creating Analytic Type Definitions](#)

[CreateOptInterface](#)

[OptBase Application Class](#)

[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 an analytic instance ID.

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

### Related Links

[ShutDown](#)

## Deleting Existing Analytic Instances

To delete an existing analytic instance for an analytic type:

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

Use the analytic instance ID as the key value to find and delete analytic instance rows from scenario-managed optimization application tables.

3. Use the function `DeleteOptProbInst` with the analytic type and analytic instance as arguments to delete the analytic instance ID from PeopleTools metadata.

---

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

---

### Related Links

[DeleteOptProbInst](#)

## Programming for Database Updates

You must plan for uncommitted database changes in your optimization PeopleCode. The PeopleSoft 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`, `PreSaveChange` and `PostSaveChange` PeopleCode events, and in `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

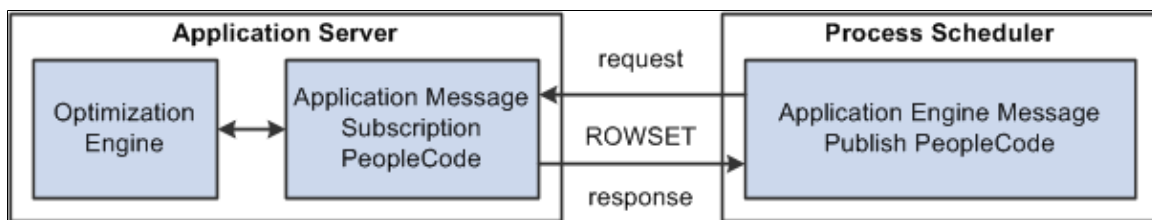
Some optimization applications can take several hours to run. These are typically run as overnight batch jobs every night when the work load is small to regenerate the optimization solution and have it ready for end users to use in the morning hence the term *lights-out mode*.

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.

### Image: Lights-out process

This diagram illustrates the 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!** 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

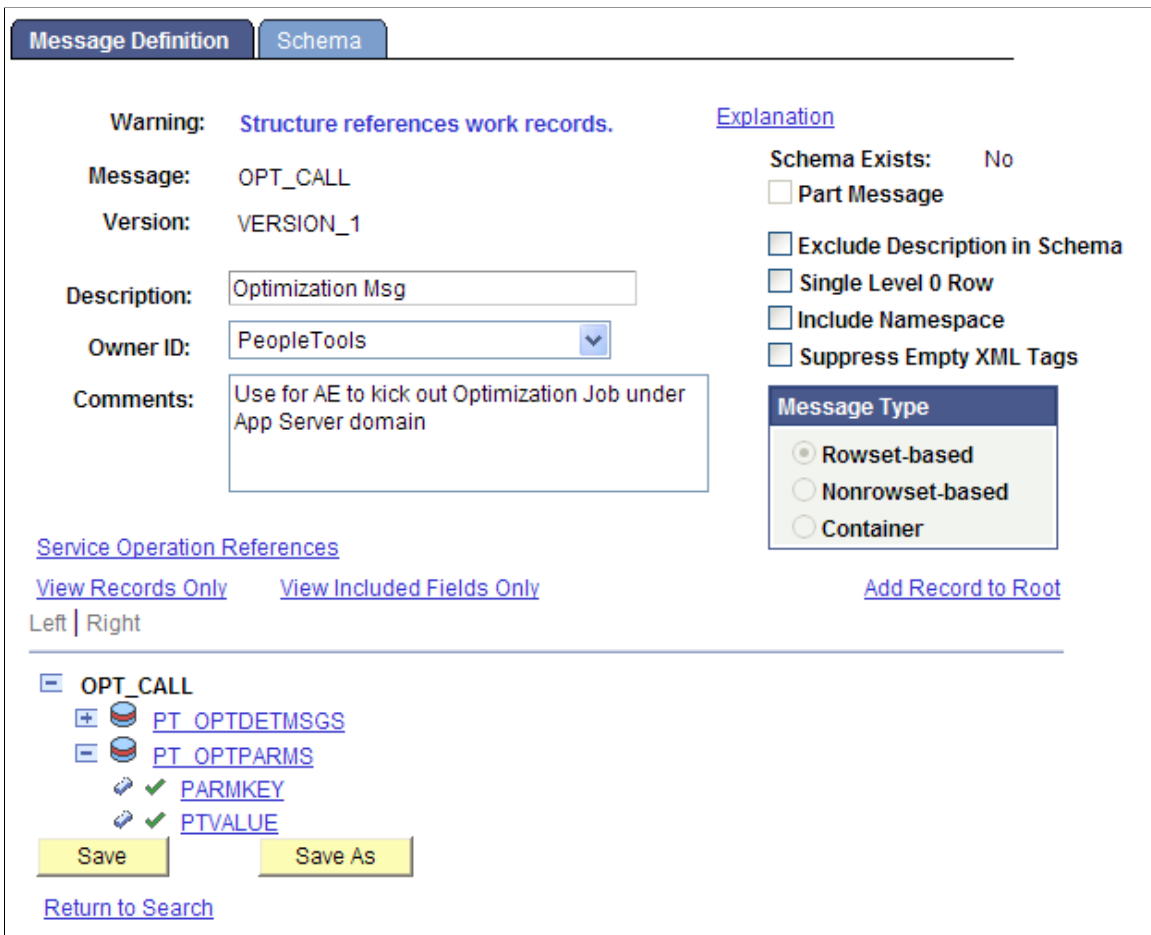
Application Engine PeopleCode in PeopleSoft Process Scheduler to and from the message OnRequest PeopleCode in the application server.

The OPT\_CALL message also uses a record, PT\_OPTDETMSGS, which contains the information needed for processing a detailed message.

This is an example of the Message Definition page (select PeopleTools, Integration Broker, Integration Setup, Messages) showing the OPT\_CALL message definition:

**Image: Message Definition page – OPT\_CALL message definition**

This example illustrates the fields and controls on the Message Definition page – OPT\_CALL message definition. You can find definitions for the fields and controls later on this page.



The OPT\_CALL message is associated with the OPT\_CALL service operation. The OPT\_CALL service operation defines the OPT\_CALL application package as a handler. This application package implements the Integration Broker methods needed to handle any messaging PeopleCode.

## 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 notification 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.

<b><i>PARMKEY Field</i></b>	<b><i>VALUE Field</i></b>
OPTCMD	CREATE  Causes the PeopleCode program implementing the Integration Broker OnRequest method to load an optimization engine. The OPT_CALL example executes the CreateOptEngine function.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.
SYNCH	<i>Y</i> if this optimization engine load is to occur synchronously, <i>N</i> 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.

<b><i>PARMKEY Field</i></b>	<b><i>VALUE Field</i></b>
OPTCMD	CHECK_STATUS  Causes the PeopleCode program implementing the Integration Broker OnRequest method to check the status of an optimization engine. The OPT_CALL example executes the CheckOptEngineStatus function.
PROBINST	The name of the analytic instance.

<b><i>PARAMKEY Field</i></b>	<b><i>VALUE Field</i></b>
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.

<b><i>PARAMKEY Field</i></b>	<b><i>VALUE Field</i></b>
OPTCMD	RUN  Causes the PeopleCode program implementing the Integration Broker OnRequest method 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 analytic 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.

<b><i>PARAMKEY Field</i></b>	<b><i>VALUE Field</i></b>
OPTCMD	SET_TRACE_LEVEL  Causes the PeopleCode program implementing the OnRequest Integration Broker method 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 analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.

<b><i>PARAMKEY Field</i></b>	<b><i>VALUE Field</i></b>
COMPONENT	<p>One of the following values:</p> <ul style="list-style-type: none"> <li>• %Opt_Engine server activity of the running optimization engine.</li> <li>• %Opt_Utility low level elements that support the running optimization engine.</li> <li>• %Opt_Datacache the in-memory database cache.</li> <li>• %Opt_Plugin the plugin being used for the running optimization engine.</li> </ul>
SEVERITY_LEVEL	<p>The severity level to log.</p> <p>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.</p> <ul style="list-style-type: none"> <li>• %Severity_Fatal</li> <li>• %Severity_Status</li> <li>• %Severity_Error</li> <li>• %Severity_Warn</li> <li>• %Severity_Info</li> <li>• %Severity_Trace1</li> <li>• %Severity_Trace2</li> </ul>

## Getting the Trace Level

To get a trace level, structure the rowset as follows, using the PT\_OPTPARMS record.

<b><i>PARAMKEY Field</i></b>	<b><i>VALUE Field</i></b>
OPTCMD	<p>GET_TRACE_LEVEL</p> <p>Causes the PeopleCode program implementing the OnRequest Integration Broker method to get the severity level at which events will be logged for an optimization engine. The OPT_CALL example executes the GetTraceLevel method.</p>
PROBINST	Set to the name of the analytic instance.

<b><i>PARAMKEY Field</i></b>	<b><i>VALUE Field</i></b>
PROCINSTANCE	Set to the name of the process instance for this process scheduler job.
COMPONENT	One of the following values: <ul style="list-style-type: none"> <li>• %Opt_Engine server activity of the running optimization engine.</li> <li>• %Opt_Utility low level elements that support the running optimization engine.</li> <li>• %Opt_Datacache the in-memory database cache.</li> <li>• %Opt_Plugin the plugin being used for the current opt engine.</li> </ul>

### Shutting Down an Optimization Engine

To shut down an optimization engine, structure the rowset as follows, using the PT\_OPTPARMS record.

<b><i>PARAMKEY Field</i></b>	<b><i>VALUE Field</i></b>
OPTCMD	SHUTDOWN  Causes the PeopleCode program implementing the OnRequest Integration Broker method to shut down an optimization engine. The OPT_CALL example executes the Shutdown method.
PROBINST	The name of the analytic 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 rowset structures in the next section (Sending Optimization Status) as an example of how to send responses from the message notification 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 PeopleCode program implementing the OnRequest Integration Broker method, 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.

<i><b>PARMKEY Field</b></i>	<i><b>VALUE Field</b></i>
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\_DETMSG record. You set key values using the PARMKEY field, and then set a value for that key field in the VALUE field.

<i><b>PARMKEY Field</b></i>	<i><b>VALUE Field</b></i>
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.
MSGPARAM1	The first parameter value.
MSGPARAM2	The second parameter value.
MSGPARAM3	The third parameter value.
MSGPARAM4	The fourth parameter value.
MSGPARAM5	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 analytic instance. In this example, the name of the analytic 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 &procinstd;
Local integer &nInst;
Local string &url;

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

Local string &stName;
Local integer &stVal;

&MSG = CreateMessage(OPERATION.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);
&procinstd = 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 = %IntBroker.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(OPERATION.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 = %IntBroker.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.MSGPARAM1).Value;
        &parm2 = &msgRec.GetField(Field.MSGPARAM2).Value;
        &parm3 = &msgRec.GetField(Field.MSGPARAM3).Value;
        &parm4 = &msgRec.GetField(Field.MSGPARAM4).Value;
        &parm5 = &msgRec.GetField(Field.MSGPARAM5).Value;
        &string = MsgGetText(&msgSet, &msgNum, "Message Not Found", &parm1,
&parm2, &parm3, &parm4, &parm5);

        End-If;
      End-For;
    End-If;
  End-If;
End-If;

```

## Editing the Response PeopleCode

The OPT\_CALL message definition 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 application package implements the Integration Broker method OnRequest. The PeopleCode in this method shows application messages for lights-out mode.

Depending upon the request message, the OnRequest method 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 OnRequest method 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 OPT\_CALL application program OnRequest method 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 = Valu⇒
e(&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.MSGPARAM1).Value = String(&arrArray [&iloop][4]);
            &msgRec.GetField(Field.MSGPARAM2).Value = String(&arrArray [&iloop][5]);
            &msgRec.GetField(Field.MSGPARAM3).Value = String(&arrArray [&iloop][6]);
            &msgRec.GetField(Field.MSGPARAM4).Value = String(&arrArray [&iloop][7]);
            &msgRec.GetField(Field.MSGPARAM5).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(analytic_inst, {%Synch | %ASynch}[, &detailedstatus] [,
processinstance])

```

## Description

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

## Parameters

### *Analytic\_inst*

Specify the analytic instance ID, which is a unique ID for this analytic instance in this optimization engine. This is supplied by users when they request that an optimization be run.

### **%Synch** | **%Asynch**

Specify whether the optimization engine is synchronous or asynchronous. The values are:

- **%Synch**: run the optimization engine synchronously.
- **%Asynch**: run the optimization engine asynchronously.

### *&detailedstatus*

Specify a variable that the engine uses to give 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\_Aiid**: The analytic 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.

---

The state record that you use with Application Engine contains the process instance ID.

See [Using Lights-Out Mode with Optimization](#).

See "Using State Records" (PeopleTools 8.54: Application Engine).

## 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 analytic 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(&additionalStatus);
if (&myInterface != NULL) then
    &status = &myInterface.ActivateModel("RMO_TEST");
    if (&status = %OptInter_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 analytic instance ID from PeopleTools metadata. This function can be called only inside FieldChange, PreSaveChange and PostSaveChange PeopleCode events, and in Workflow.

---

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

---

### Parameters

<i>probinst</i>	Enter the analytic instance ID to delete.
<i>&amp;detailedstatus</i>	(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: <ul style="list-style-type: none"> <li>• %OptEng_Success: The function completed successfully.</li> <li>• %OptEng_Fail: The function failed.</li> <li>• %OptEng_Invalid_Piid: The analytic instance ID passed to the function is invalid.</li> <li>• %OptEng_Sql_Exception: A SQLException is encountered when access database.</li> </ul>

- %OptEng\_Exists: An analytic server loaded with this analytic instance still exists.

## Returns

Returns %OptEng\_Success if successful; otherwise returns %OptEng\_Fail.

## Example

The following example deletes the instance for an analytic type:

---

**Note:** Whenever you add records to an analytic type, you must call DeleteOptProbInst to delete the old analytic 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 analytic instance " | &probinst | "
    failed.";
Else
    QEOPT_WRK.MESSAGE_TEXT = "Analytic 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 analytic instance " | &probinst | " failed
    for bad piid.";
Else
    QEOPT_WRK.MESSAGE_TEXT = "Analytic 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 analytic instance.

---

**Note:** You cannot call GetOptEngine from a domain other than the application server.

---

### Parameters

<i>probinst</i>	Enter the analytic instance ID, which is unique ID for this analytic 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 analytic 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 analytic instance:

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

&myopt = GetOptEngine(&probinst);
If &myopt <> NULL then
&status = &myopt.ShutDown();
QEOPT WRK.MESSAGE TEXT = "Analytic 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 analytic instance IDs in an analytic type.

**Parameters*****ProblemType***

Enter the name of the analytic type that you created in 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 analytic instance IDs that start with this

prefix. If not used, all the analytic instance IDs in the analytic 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 analytic type name passed to the function is invalid.

## **Returns**

Returns an array of strings containing the optimization analytic 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, &ret, &detailedstatus);

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

## **InsertOptProbInst**

### **Syntax**

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

### **Description**

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



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

---

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

---

## Parameters

<i><b>probinst</b></i>	Enter the analytic instance ID to be inserted into the analytic type.
<i><b>ProblemType</b></i>	Enter the name of the analytic type that you created in Application Designer.
<i><b>&amp;detailedstatus</b></i>	(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: <ul style="list-style-type: none"> <li>• <code>%OptEng_Success</code>: The function completed successfully.</li> <li>• <code>%OptEng_Fail</code>: The function failed.</li> <li>• <code>%OptEng_Invalid_Piid</code>: The analytic instance ID passed to the function is invalid.</li> </ul>
<i><b>Description</b></i>	(Optional) Specify a description for the analytic instance. This parameter takes a string value.

## Returns

This method returns a constant. Valid values are:

<b>Value</b>	<b>Description</b>
<code>%OptEng_Success</code>	Returned if method succeeds.
<code>%OptEng_Fail</code>	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 analytic instance "
    | &probinst | " failed.";
Else
    QEOPT_WRK.MESSAGE_TEXT = "Analytic 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 analytic instance "
    | &probinst | " failed for bad piid.";
End-if;
```

## IsValidOptProbInst

### Syntax

```
IsValidOptProbInst(probinst [, &detailedstatus])
```

### Description

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

### Parameters

- |                            |  |
|----------------------------|--|
| <i>probinst</i>            | Enter the analytic instance ID to be validated.  |
| <i>&amp;detailedstatus</i> | (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: <ul style="list-style-type: none"> <li>• %OptEng_Success: The function completed successfully.</li> <li>• %OptEng_Invalid_Piid: The analytic type name passed to the function is invalid.</li> </ul> |

### Returns

This method returns a constant. Valid values are:

<b>Value</b>	<b>Description</b>
%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
- %OptEng\_Not\_Available

---

**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.

<i><b>Numeric Value</b></i>	<i><b>Constant Value</b></i>	<i><b>Description</b></i>
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.
23	%OptEng_Idle	The optimization engine is loaded with an application problem and waiting for a transaction request.

<b>Numeric Value</b>	<b>Constant Value</b>	<b>Description</b>
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.
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.

- %OptEng\_Wrong\_Parm\_Type

## Parameters

### *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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

### *&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 analytic 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:

<i>Value</i>	<i>Description</i>
%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

#### *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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

## Returns

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

## Example

See [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

***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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

### Returns

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

### Example

See [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

*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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

## Returns

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

## Example

See [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

*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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

## Returns

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

## Example

See [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

*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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

## 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

*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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

## Returns

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

## Example

See [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

*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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

## Returns

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

## Example

See [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

***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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

## 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

***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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

## Returns

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

## Example

See [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

***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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

### Returns

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

### Example

See [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

***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, PARAM_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

***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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

### Returns

This method returns a constant. Valid values are:

<b><i>Value</i></b>	<b><i>Description</i></b>
%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*, *PARAM\_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

**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 analytic type definition.

See [Configuring Analytic Type Transactions](#).

## Returns

This method returns a constant. Valid values are:

<b>Value</b>	<b>Description</b>
%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

### *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:

<b>Value</b>	<b>Description</b>
%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 Application Engine.

---

## Parameters

None.

## Returns

This method returns a constant. Valid values are:

<b>Value</b>	<b>Description</b>
%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 an 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 an analytic type definition. These methods apply to the parameters that are defined for the transaction in the analytic 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 analytic 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 analytic 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 analytic 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 analytic 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 [Configuring Analytic Type Records](#).

---

## 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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
& <i>parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
& <i>parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
& <i>parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
& <i>parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
& <i>parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
& <i>parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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 an analytic 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 analytic 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

***&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 analytic 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

***&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 analytic 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

<i>&amp;OldRecord</i>	Specifies a record variable that contains the pre-update content of the data row that was updated.
<i>&amp;NewRecord</i>	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 analytic 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

<i>&amp;OldRecord</i>	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 analytic 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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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

<i>parmName</i>	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
<i>&amp;parmVal</i>	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 analytic 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 [DeactivateModel](#).

### Parameters

#### *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 analytic 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 analytic type definition.

### Returns

This method returns a constant value. Valid values are:

<b>Value</b>	<b>Description</b>
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the solver fails to solve the problem.

### Example

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

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

## ActivateObjective

### Syntax

```
ActivateObjective(Model_Name, Objective_Name)
```

## Description

Use the `ActivateObjective` method to activate the specified objective for an optimization model.

## Parameters

<i>Model_Name</i>	Specify the name of the model.
<i>Objective_Name</i>	Specify the name of the objective.

## Returns

This method returns a constant value. Valid values are:

<b>Value</b>	<b>Description</b>
<code>%OptInter_Success</code>	Returned if method succeeds.
<code>%OptInter_Fail</code>	Returned if the solver fails to solve the problem.

## DeactivateModel

### Syntax

```
DeactivateModel (ModelID)
```

### Description

The `DeactivateModel` method detaches the solver from the specified model.

### Parameters

<i>ModelID</i>	Specify the name of the optimization model you want to deactivate. This must be the name of one of the models associated with the analytic type definition.
----------------	---

### Returns

This method returns a constant value. Valid values are:

<b>Value</b>	<b>Description</b>
<code>%OptInter_Success</code>	Returned if method succeeds.
<code>%OptInter_Fail</code>	Returned if the solver fails to solve the problem.

### 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

<i>LogSeverity</i>	Specify the severity level of the message, as one of the following system constants: <ul style="list-style-type: none"> <li>• %Severity_Fatal</li> <li>• %Severity_Status</li> <li>• %Severity_Error</li> <li>• %Severity_Warn</li> <li>• %Severity_Info</li> <li>• %Severity_Trace1</li> <li>• %Severity_Trace2</li> </ul>
<i>Message</i>	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

<i><b>&amp;Record</b></i>	Specify a record with the same structure as the records that comprise the rowset, with its key fields populated.
<i><b>startrow</b></i>	Specify as an integer the starting row number of the search. Specify 0 to search from the first row in the rowset.
<i><b>endrow</b></i>	Specify as an integer the ending row number of the search. Specify 0 to search through the last row in the rowset.
<i><b>fieldname</b></i>	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

### *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:

<b>Value</b>	<b>Description</b>
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the solver fails to solve the problem.

## 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

#### *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:

<b>Value</b>	<b>Description</b>
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if solver fails to solve the problem.

## 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 Constraint "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

*ModelID* Specify the model ID as a string. This is the name used for the model definition in 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

<i>modelID</i>	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 Application Designer.
<i>varArrayID</i>	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

<i>modelID</i>	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.
<i>varArrayID</i>	Specify as a string the name of the variable array being optimized. Your application documentation should provide this name.
<i>boundType</i>	Specify a system constant indicating which bounding values to override. The value you specify here determines how the <i>lowerBound</i> and <i>upperBound</i> parameters are applied to the specified model. <ul style="list-style-type: none"> <li><code>%OPT_LOWER_BOUND</code>: Override only the lower bound as specified by the <i>lowerBound</i> parameter. The <i>upperBound</i> parameter is ignored.</li> <li><code>%OPT_UPPER_BOUND</code>: Override only the upper bound as specified by the <i>upperBound</i> parameter. The <i>lowerBound</i> parameter is ignored.</li> </ul>



- `%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 analytic 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

#### *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 analytic 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

#### *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 analytic type definition. You can override any or all of the solver setting parameter values.

See [Configuring Models for Optimization](#).

### Returns

One of the following system constants:

%OptInter\_Fail: The solver fails to solve the problem.

%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");
```

# Administering Optimization Server Components

---

## Administering Optimization Server Components

An analytic server is a type of PeopleSoft application server. An optimization engine is an analytic server loaded with an optimization analytic instance. You administer optimization engines using the standard application server tools.

### Related Links

"System and Server Administration" (PeopleTools 8.54: System and Server Administration)

