PeopleSoft®

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Developer's Guide

June 2004

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Developer's Guide
SKU PT845PCD-B 0604

Open Source Disclosure

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

Apache Software Foundation

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

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

OpenSSL

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

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

SSLeay

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

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

Loki Library

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

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

Contents

About This PeopleBook

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

This preface discusses:

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

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

PeopleSoft Application Prerequisites

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

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

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

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

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

PeopleSoft Application Fundamentals

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

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

Related Documentation

This section discusses how to:

- Obtain documentation updates.
- Order printed documentation.

Obtaining Documentation Updates

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

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

See Also

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

Ordering Printed Documentation

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

- Web
- Telephone
- Email

Web

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

Telephone

Contact MMA Partners at 877 588 2525.

Email

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

See Also

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

Typographical Conventions and Visual Cues

This section discusses:

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

Typographical Conventions

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

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

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

Visual Cues

PeopleBooks contain the following visual cues.

Notes

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

Note. Example of a note.

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

Important! Example of an important note.

Warnings

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

Warning! Example of a warning.

Cross-References

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

Country, Region, and Industry Identifiers

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

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

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

Country Identifiers

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

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

Region Identifiers

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

- Asia Pacific
- Europe
- Latin America
- North America

Industry Identifiers

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

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

Currency Codes

Monetary amounts are identified by the ISO currency code.

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

Comments and Suggestions

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

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

Or send email comments to doc@peoplesoft.com.

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

Common Elements in These PeopleBooks

As of Date The last date for which a report or process includes data.

Business Unit An ID that represents a high-level organization of business information. You can use a business unit to define regional or departmental units within a larger organization.

Description Enter up to 30 characters of text.

Effective Date The date on which a table row becomes effective; the date that an action begins. For example, to close out a ledger on June 30, the effective date for the ledger closing would be July 1. This date also determines when

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

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Process Scheduler

Enterprise PeopleTools 8.45 PeopleBook: Using PeopleSoft Applications

PeopleCode Developer's Guide Preface

This preface includes typographical conventions used in PeopleCode and discusses the uses of this book.

PeopleCode Developer's Guide

This PeopleBook covers the concepts of PeopleCode, the programming language used in the development of PeopleSoft applications. Its chapters describe techniques for adding PeopleCode to applications, tips for using PeopleCode, the interaction of PeopleCode and the Component Processor, and a number of other specialized topics, such as the use of the PeopleCode debugger and referencing data in the component buffer.

The accompanying book, PeopleCode Reference, is a complete reference of the PeopleCode language. Its chapters describe the syntax and fundamental elements of the PeopleCode language.

The "About These PeopleBooks" preface contains general product line information, such as related documentation, common page elements, and typographical conventions. This preface also contains a glossary with useful terms that are used in PeopleBooks.

PeopleCode Typographical Conventions

Throughout this book, we use typographical conventions to distinguish between different elements of the PeopleCode language, such as bold to indicate function names, italics for arguments, and so on.

Please take a moment to review the following typographical cues:

Font Type	Description
`monospace font`	Indicates a PeopleCode program or other example
Keyword	In PeopleCode syntax, items in **keyword** font indicate function names, method names, language constructs, and PeopleCode reserved words that must be included literally in the function call.
Variable	In PeopleCode syntax, items in *variable* font are placeholders for arguments that your program must supply.
...	In PeopleCode syntax, ellipses indicate that the preceding item or series can be repeated any number of times.

Font Type	Description
{*Option1*|*Option2*}	In PeopleCode syntax, when there is a choice between two options, the options are enclosed in curly braces and separated by a pipe.
[ ]	In PeopleCode syntax optional items are enclosed in square brackets.
&Parameter	In PeopleCode syntax an ampersand before a parameter indicates that the parameter is an already instantiated object.

CHAPTER 1

Getting Started with PeopleCode

PeopleCode is the propriatary language used by PeopleSoft applications. This chapter provides an overview of PeopleCode and discusses how to create PeopleCode programs.

This chapter provides information to consider before you begin to use PeopleCode. In addition to the considerations presented in this section, you should take advantage of all PeopleSoft sources of information, including the installation guides, release notes, and PeopleBooks.

PeopleCode Overview

This section provides an overview of the conceptual information available about the PeopleCode language. The reference material, that is, the actual descriptions of the functions, methods and properties can be found in the following:

- *PeopleTools 8.45 PeopleBook: PeopleCode Language Reference*

  This book contains information about PeopleCode built-in functions, meta-SQL, system variables, and meta-HTML.

- *PeopleTools 8.45 PeopleBook: PeopleCode API Reference*

  This book contains information about all the classes delivered with PeopleTools, as well as specifics about each class's methods and properties.

PeopleCode resembles other programming languages. However, many aspects are unique to the language and the PeopleTools environment. To learn more about the language, see Understanding the PeopleCode Language.

See Chapter 2, "Understanding the PeopleCode Language," page 5.

PeopleCode is an object-oriented language. To learn about objects and how they're used in PeopleCode, see Understanding Objects and Classes in PeopleCode.

See Chapter 3, "Understanding Objects and Classes in PeopleCode," page 35.

The component buffer is the area in memory that stores data for the currently active component. Which fields are loaded into the component buffer, as well as how to access them, is covered in Referencing Data in the Component Buffer.

See Chapter 4, "Referencing Data in the Component Buffer," page 41.

The system uses a data buffer as well as the component buffer. The data buffer is used to store data added from sources other than the component, such as from a PeopleSoft Application Engine program, a message, and so on. For information about this buffer, see Accessing the Data Buffer.

See Chapter 5, "Accessing the Data Buffer," page 59.

All PeopleCode is associated with a definition and an event. The events run in a particular order from the Component Processor. To learn more about the Component Processor and the standard event set, see PeopleCode and the Component Processor.

See Chapter 6, "PeopleCode and the Component Processor," page 81.

There are considerations you should take into account when creating applications to be used in the PeopleSoft Pure Internet Architecture. These include how to make your code more efficient when running on the internet, as well as considerations when using specific definition. For more information, see PeopleCode and the PeopleSoft Pure Internet Architecture.

See Chapter 7, "PeopleCode and PeopleSoft Pure Internet Architecture," page 127.

There are restrictions on using some of the functions and methods in the PeopleCode language, as well as considerations for others, like using standalone rowsets and the OLE functions. These are covered in the Using Methods and Built-in Functions chapter.

See Chapter 8, "Using Methods and Built-In Functions," page 135.

PeopleCode has a tremendous amount of specialized functionality, such as:

- using the GenerateTree function to create a tree in your application.
- viewing, adding or deleting files.
- creating applications that run on mobile devices, such as PDAs or laptops.

See Chapter 9, "Understanding HTML Trees and the GenerateHTML Function," page 165.

See Chapter 10, "Understanding File Attachments and PeopleCode," page 183.

See Chapter 11, "Understanding Mobile Pages and Classes," page 193.

Creating PeopleCode Programs

All PeopleCode programs are associated with a definition as well as an event. To learn more about where you can place your PeopleCode, and have it executed as part of the Component Processor event flow, see Accessing PeopleCode and Events.

See Chapter 12, "Accessing PeopleCode and Events," page 205.

Use the PeopleCode editor to create your PeopleCode programs. All the functionality of the PeopleCode editor is described in Using the PeopleCode Editor.

See Chapter 13, "Using the PeopleCode Editor," page 219.

Every PeopleCode program is associated with a definition. The following definitions have additional functionality associated with the PeopleCode editor:

- SQL definitions
- Application Package definitions

See Chapter 14, "Using the SQL Editor," page 237.

See Chapter 15, "Creating Application Packages and Classes," page 245.

After you've created your program, you must run it, and often, that also involves fixing any errors that you find. The PeopleCode debugger is an integrated part of PeopleSoft Application Designer, and has many useful tools for determining where code errors are occurring. All the functionality is described in Debugging your Application.

See Chapter 16, "Debugging Your Application," page 251.

After your PeopleCode program is running, you may want to either improve its performance or the user experience. Techniques for doing this are discussed in Improving your PeopleCode.

See Chapter 17, "Improving Your PeopleCode," page 275.

CHAPTER 2

Understanding the PeopleCode Language

This chapter discusses:

- PeopleCode language structure.
- Data types.
- Statements.
- Functions.
- Expressions.
- Operators.

PeopleCode Language Structure

This chapter assumes that you are familiar with a programming language, such as C, Visual Basic, or Java.

In its fundamentals, PeopleCode syntax resembles other programming languages. Some aspects of the PeopleCode language, however, are specifically related to the PeopleTools environment. Definition name references, for example, enable you to refer to PeopleTools definitions, such as record definitions or pages, without using hard-coded string literals. Other language features, such as PeopleCode data types and metastrings, reflect the close interaction of PeopleTools and Structured Query Language (SQL). Dot notation, classes and methods in PeopleCode are similar to other object oriented languages, like Java.

Data Types

Conventional data types include number, date, string. Use them for basic computing. Object data types instantiate objects from PeopleTools classes. The appropriate used of each data type is demonstrated where the documentation discusses PeopleCode that uses that data type.

Declare variables before you use them.

This section discusses:

- Conventional data types.
- Object data types.

See Also

Chapter 2, “Understanding the PeopleCode Language,” Variables, page 23

Conventional Data Types

PeopleCode includes these conventional data types:

- Any

  When variables and function return values are declared as Any, the data type is indeterminate, enabling PeopleTools to determine the appropriate type of value based on context. Undeclared local variables are Any by default.

- Boolean
- Date
- DateTime
- Float
- Integer

Note. The Float and Integer data types should be used instead of Number only where a performance analysis indicates that the increased speed is useful and an application analysis indicates that the different representations won't affect the results of the computations.

- Number
- Object
- String
- Time

Considerations for Float, Integer, and Number Types

The Integer type is a number represented as a 32-bit signed twos complement number, so it has a range of -2,147,483,648 to 2,147,483,647.

The Float type is a number represented using the machine floating binary point (double precision) representation. This floating binary point representation is not appropriate for exact calculations involving decimal fractions; in particular, calculations involving money. For example, because a tenth (1/10 or .1) cannot be exactly represented in floating binary point, a floating binary point sum of .10 + .10 is not be equal to .20. The default number representation in PeopleCode is a floating decimal point representation, which avoids this difficulty.

Operations (other than division) are done using integer arithmetic if the operands are both integers and the destination is an integer, even if the variable is declared as the Number type. The destination is considered to be an integer if one of the following is true:

- The destination is an assignment to an integer variable or parameter.
- The destination is an array subscript.
- The destination is the right-hand operand of a comparison and the left-hand operand is an integer.
- The destination is a when-expression part of an evaluate statement, and the expression evaluated at the start of the evaluate statement is an integer.
- The destination is a for-loop initial, limit, or step expression and the control variable of the for-loop is an integer.

Division (the / operator) is never performed using integer arithmetic. It's always performed using the floating-decimal-point arithmetic, even if the result variable is declared as an Integer type.

Follow these recommendations for assigning types to numbers:

- Use Number for most application data values.
- Use Integer when you are counting items, such as rows in a rowset.
- Use Float only when you are tuning the code for performance (after it is already working).

  In addition, you should only use the Float type when you are certain that the resulting loss of precision will not affect the application and that the increase in the speed of the computation makes a difference to the transaction. In general, few applications should use the Float type.

Data Types

For most classes in PeopleTools, you need a corresponding data type to instantiate objects from that class.

See Chapter 3, "Understanding Objects and Classes in PeopleCode," page 35.

PeopleCode includes these data buffer access types:

- Field
- Record
- Row
- Rowset

PeopleCode includes these display data types:

- Chart
- Grid
- GridColumn
- Page

PeopleCode includes these internet script data types:

- Cookie
- Request
- Response

PeopleCode includes these miscellaneous data types:

- AESection
- Array
- Crypt
- Exception
- File
- Interlink
- BIDocs

Note. BIDocs and Interlink objects used in PeopleCode programs run on the application server can only be declared as type Local. You can declare Interlinks as Global only in an Application Engine program.

See Chapter 2, "Understanding the PeopleCode Language," User-Defined Variable Declaration and Scope, page 24.

- JavaObject

Note. JavaObject objects can only be declared as type Local.

- Message
- MCFIMInfo
- OptEngine
- PostReport
- ProcessRequest
- RowsetCache
- SoapDoc
- SQL
- SyncServer
- TransformData

Note. TransformData objects can only be declared as type Local.

- XmlDoc
- XmlNode

Note. XmlNode objects can only be declared as type Local.

API Object Types

Use this data type for any API object, such as a session object, a tree object, a component interface, a portal registry, and so on.

The following ApiObject data type objects can be declared as type Global:

- Session
- PSMessages collection
- PSMessages
- All tree classes (trees, tree structures, nodes, levels, and so on.)
- All query classes

All other ApiObject data type objects must be declared as Local.

Comments

Use comments to explain, preferably in language comprehensible to anyone reading your program, what your code does. Comments also enable you to differentiate between PeopleCode delivered with the product and PeopleCode that you add or change. This differentiation helps in your analysis for debugging and upgrades.

Note. Use comments to place a unique identifier marking any changes or enhancements that you have made to a PeopleSoft application. This makes it possible for you to search for all the changes you have made. This is particularly helpful when you are upgrading a database.

There are three ways to insert comments into PeopleCode.You can surround comments with /* at the beginning and */ at the end. You can also use a REM (remark) statement for commenting. Put a semicolon at the end of a REM comment. If you do not, everything up to the end of the next statement is treated as part of the comment.

Note. Commented text cannot exceed a maximum of 16383 characters.

To enclose one set of comments within a another set, use the symbol <* at the start and *> at the end. You generally use this when you're testing code and want to comment out a section that already contains comments. The following code sample shows comment formatting:

```
<* this program is no longer valid commenting out
entire thing

REM This is an example of commenting PeopleCode;
/* ----- Logic for Compensation Change ----- */
/* Recalculate compensation change for next row.
Next row is based on prior value of EFFDT.  */

calc_next_compchg(&OLDDT, EFFSEQ, 0);

/* Recalculate compensation change for current row and next row.
Next row is based on new value of EFFDT.  */

calc_comp_change(EFFDT, EFFSEQ, COMP_FREQUENCY, COMPRATE,
CHANGE_AMT, CHANGE_PCT);

calc_next_compchg(EFFDT, EFFSEQ, 0);

*>
```

Note. All text between the <* and *> comment markers is scanned. If you have mismatched quotation marks, invalid assignments, and so on, you may receive an error when using these comments.

Statements

A statement can be a declaration, an assignment, a program construct (such as a Break statement or a conditional loop), or a subroutine call.

This section discusses:

- Separators.
- Assignment statements.
- Language constructs.
- Branching statements.

- Conditional loops.

Separators

PeopleCode statements are generally terminated with a semicolon. The PeopleCode language accepts semicolons even if they're not required, such as after the last statement completed within an If statement. This enables you to consistently add semicolons after each statement.

Extra spaces are ignored. They are removed by the PeopleCode Editor when you save the code.

Assignment Statements

The assignment statement is the most basic type of statement in PeopleCode. It consists of an equal sign with a variable name on the left and an expression on the right:

```
variableName = expression;
```

The expression on the right is evaluated, and the result is placed in the variable named on the left. Depending on the data types involved, the assignment is passed either by value or by reference.

Assignment by Value

In most types of assignments, the result of the right-hand expression is assigned to the variable as a newly created value, in the variable's own allocated memory area. Subsequent changes to the value of that variable have no effect on any other data.

Assignment by Reference

When both sides of an assignment statement are object variables, the result of the assignment is not to create a copy of the object in a unique memory location and assign it to the variable. Instead, the variable points to the object's memory location. Additional variables can point to the same object location.

For example, both &AN and &AN2 are arrays of type Number. Assigning &AN2 to &AN does *not* assign a copy of &AN2 to &AN. Both array objects point to the same information in memory.

```
Local array of number &AN, &AN2;
Local number &NUM;

&AN = CreateArray(100, 200, 300);
&AN2 = &AN;
&NUM = &AN[1];
```

In the code example, &AN2 and &AN point to the same object: an array of three numbers. If you were to change the value of &AN[2] to 500, and then reference the value of &AN2[2], you would get 500, not 300. On the other hand, assigning &NUM to the first element in &AN (100), is *not* an object assignment. It's an assignment by value. If you changed &AN[1] to 500, then &NUM remains 200.

Note. In PeopleCode, the equal sign can function as either an assignment operator or a comparison operator, depending on context.

Language Constructs

PeopleCode language constructs include:

- Branching structures: If and Evaluate.
- Loops and conditional loops: For, Repeat, and While.
- Break, Continue, and Exit statements loop control and terminating programs.
- The Return statement for returning from functions.
- Variable and function declaration statements: Global, Local, and Component for variables, and Declare Function for functions.
- The Function statement for defining functions.
- Class definition statements.
- Try, Catch, and Throw statements for error handling.

Functions as Subroutines

PeopleCode, like C, does not have subroutines as we generally refer to them. PeopleCode subroutines are the subset of PeopleCode functions only that are defined to return no value or to return a value optionally. Calling a subroutine is the same as calling a function with no return value:

```
function_name([param_list]);
```

See Also

Chapter 2, "Understanding the PeopleCode Language," Branching Statements, page 11

Chapter 2, "Understanding the PeopleCode Language," Functions, page 14

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

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

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

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

Branching Statements

Branching statements control program flow based on evaluation of conditional expressions.

If, Then, and Else statements

The syntax of If, Then, and Else statements is:

```
If condition Then
   [statement_list_1]
[Else
   [statement_list_2]]
End-if;
```

This statement evaluates the Boolean expression *condition*. If *condition* is true, the If statement executes the statements in *statement_list_1*. If *condition* is false, then the program executes the statements in the Else clause; if there is no Else clause, the program continues on to the next statement.

Evaluate Statement

Use the Evaluate statement to check multiple conditions. Its syntax is:

```
Evaluate left_term
   When [relop_1] right_term_1
     [statement_list]
   .
   .
   .
  When [relop_n] right_term_n
     [statement_list]
   [When-other
      [statement_list]]
End-evaluate;
```

The Evaluate statement takes an expression, *left_term*, and compares it to compatible expressions *(right_term)* using the relational operators *(relop)* in a sequence of When clauses. If *relop* is omitted, then the equal sign is assumed. If the result of the comparison is TRUE, the program executes the statements in the When clause, then moves on to evaluate the comparison in the following When clause. The program executes the statements in all of the When clauses for which the comparison evaluates to TRUE. If none of the When comparisons evaluates to True, the program executes the statement in the When-other clause, if one is provided. For example, the following Evaluate statement executes only the first When clause. &USE_FREQUENCY in the following example can only have one of three string values:

```
evaluate &USE_FREQUENCY
when = "never"
   PROD_USE_FREQ = 0;
when = "sometimes"
   PROD_USE_FREQ = 1;
when = "frequently"
   PROD_USE_FREQ = 2;
when-other
   Error "Unexpected value assigned to &USE_FREQUENCY."
end-evaluate;
```

To end the Evaluate statement after the execution of a When clause, you can add a Break statement at the end of the clause, as in the following example:

```
evaluate &USE_FREQUENCY
when = "never"
   PROD_USE_FREQ = 0;
   Break;
when = "sometimes"
   PROD_USE_FREQ = 1;
   Break;
when = "frequently"
   PROD_USE_FREQ = 2;
   Break;
when-other
```

```
      Error "Unexpected value assigned to &USE_FREQUENCY."
   end-evaluate;
```

In rare cases you may want to make it possible for more than one When clause to execute, as shown in the following example:

```
   evaluate &PURCHASE_AMT
   when >= 100000
      BASE_DISCOUNT = "Y";
   when >= 250000
      SPECIAL_SERVICES = "Y";
   when >= 1000000
      MUST_GROVEL = "Y";
   end-evaluate;
```

For Statement

The For statement repeats a sequence of statements a specified number of times. Its syntax is:

```
   For count = expression1 to expression2
       [Step i];
       statement_list
   End-for;
```

The For statement initializes the value of *count* to *expression1*, then increments *count* by *i* each time after it executes the statements in *statement_list.* The program continues in this loop until *count* is equal to *expression2.* If the Step clause is omitted, then *i* equals one. To count backwards from a higher value to a lower value, then use a negative value for *i.* You can exit a For loop using a Break statement.

The following example demonstrates the For statement:

```
   &MAX = 10;
   for &COUNT = 1 to &MAX;
      WinMessage("Executing statement list, count = " | &COUNT);
   end-for;
```

See Also

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

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

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

Conditional Loops

Conditional loops, Repeat and While, repeat a sequence of statements, evaluating a conditional expression each time through the loop. The loop terminates when the condition evaluates to True. You can exit from a conditional loop using a Break statement. If the Break statement is in a loop embedded in another loop, the break applies only to the inside loop.

Repeat Statement

The syntax of the Repeat statement is:

```
Repeat
  statement_list
Until logical_expression;
```

The Repeat statement executes the statements in *statement_list* once, then evaluates *logical_expression*. If *logical_expression* is false, the sequence of statements is repeated until *logical_expression* is true.

While Statement

The syntax of the While statement is:

```
While logical_expression
     statement_list
End-while;
```

The While statement evaluates *logical_expression* before executing the statements in *statement_list.* It continues to repeat the sequence of statements until *logical_expression* evaluates to False.

See Also

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

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

Functions

This section discusses:

- Supported functions.
- Function definitions.
- Function declarations.
- Function calls.
- Function return values.
- Function naming conflicts.

See Also

Chapter 3, "Understanding Objects and Classes in PeopleCode," page 35

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

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

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

Supported Functions

PeopleCode supports the following types of functions:

- Built-in: The standard set of PeopleCode functions. These can be called without being declared.
- Internal: Functions that are defined (using the Function statement) withint the PeopleCode program in which they are called.
- External PeopleCode: PeopleCode functions defined outside the calling program. These are generally contained in record definitions that serve as function libraries.
- External non-PeopleCode: Functions stored in external (C-callable) libraries.

In addition, PeopleCode supports methods. The main differences between a built-in function and a method are:

- A built-in function, in your code, is on a line by itself, and does not (generally) have any dependencies.

  You do not have to instantiate an object before you can use the function.
- A method can only be executed by an object (using dot notation).

  You have to instantiate the object first.

Function Definitions

PeopleCode functions can be defined in any PeopleCode program. Function definitions must be placed at the top of the program, along with any variable and external function declarations.

By convention, PeopleCode programs are stored in records whose names begin in FUNCLIB_, and they are always attached to the FieldFormula event.

Note. Application classes can provide an alternative, and sometimes cleaner, mechanism for separating out functionality than functions stored in FUNCLIBs.

See Also

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

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Application Classes"

Function Declarations

If you call an external function from a PeopleCode program, you must declare the function at the top of the program. The syntax of the function declaration varies, depending on whether the external function is written in PeopleCode or compiled in a dynamic link library.

The following is an example of a function declaration of a function that is in another FUNCLIB record definition:

```
Declare Function UpdatePSLOCK PeopleCode FUNCLIB_NODES.MSGNODENAME FieldFormula;
```

See Also

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

Function Calls

Functions are called with this syntax:

```
function_name([param_list])
```

The parameter list *(param_list)*,is a list of expressions, separated by commas, that the function expects you to supply. Items in the parameter list can be optional or required.

You can check the values of parameters that get passed to functions at runtime in the Parameter window of the PeopleCode debugger.

If the return value is required, then the function must be called as an expression, for example:

```
&RESULT = Product(&RAISE_PERCENT, .01, EMPL_SALARY);
```

If the function has an optional return value, it can be called as a subroutine. If the function has no return value, it *must* be called as a subroutine:

```
WinMessage(64, "I can't do that, " | &OPER_NICKNAME | ".");
```

Parameters are always passed to internal and external PeopleCode functions by reference. If the function is supposed to change the data the caller passes, you must also pass in a variable.

Built-in function parameters can be passed by reference or by value, depending on the function. External C function parameters can be passed by value or by reference, depending on the declaration and type.

See Also

Chapter 16, "Debugging Your Application," page 251

Chapter 2, "Understanding the PeopleCode Language," Variables and Functions, page 28

Function Return Values

Functions can return values of any supported data type; some functions do not return any value.

Optional return values occur only in built-in functions. You cannot define a function that optionally returns a value. Optional return values are typical in functions that return a Boolean value indicating whether execution was successful. For example, the following call to DeleteRow ignores the Boolean return value and deletes a row:

```
DeleteRow(RECORD.BUS_EXPENSE_PER, &L1_ROW, RECORD.BUS_EXPENSE_DTL, &L2_ROW);
```

The following example checks the return value and returns a message saying whether it succeeded:

```
if DeleteRow(RECORD.BUS_EXPENSE_PER, &L1_ROW, RECORD.BUS_EXPENSE_DTL, &L2_ROW) then
   WinMessage("Row deleted.");
else
   WinMessage("Sorry -- couldn't delete that row.");
end-if;
```

Function Naming Conflicts

If you define a function with the same name as a built-in function, the function that you defined takes precedence over the built-in function.

Anytime you compile the PeopleCode in the PeopleCode Editor, a warning message appears in the Validate tab, indicating that a user-defined function has the same name as an existing built-in function.

In addition, if you select Compile All PeopleCode, an error message is generated in the log file for every user-defined function that has the same name as a built-in function.

The following is an example error message: User defined function IsNumber is overriding the builtin function of the same name. (2,98)

If you notice that you named a function the same as a built-in function, and that the built-in function does what you're trying to achieve, replace your function with a reference to the built-in function.The built-in function is probably more efficient. In addition, using the built-in function reduces confusion for people who maintain your code, because if they miss the warning message in the Validate tab, they might assume the built-in function is being called when it is not.

Expressions

This section discusses:

- Expression fundamentals.
- Constants.
- Functions as expressions.
- System variables.
- Metastrings.
- Record field references.
- Definition name references.

See Also

Chapter 2, "Understanding the PeopleCode Language," Variables, page 23

Expression Fundamentals

Expressions evaluate to values of PeopleCode data types. A simple PeopleCode expression can consist of a constant, a temporary variable, a system variable, a record field reference, or a function call. Simple expressions can be modified by unary operators (such as a negative sign or logical NOT), or combined into compound expressions using binary operators (such a plus sign or logical AND).

Definition name references evaluate to strings equal to the name of a PeopleTools definition, such as a record, page, or message. They enable you to refer to definitions without using string literals, which are difficult to maintain.

Metastrings (also called meta-SQL) are special expressions used within SQL string literals. At runtime, the metastrings expand into the appropriate SQL for the current database platform.

Constants

PeopleCode supports numeric, string, and Boolean constants, as well as user-defined constants. It also supports the constant Null, which indicates an object reference that does not refer to a valid object.

Note. You can express Date, DateTime, and Time values by converting from String and Number constants using the Date, Date3, DateTime6, DateTimeValue, DateValue, Time3, TimePart, and the TimeValue functions. You can also format a DateTime value as text using FormatDateTime.

Numeric Constants

Numeric constants can be any decimal number. Some examples are:

- 7
- 0.8725
- -172.0036

String Constants

String constants can be delimited by using either single (’) or double (") quotation marks. If a quotation mark occurs as part of a string, the string can be surrounded by the other delimiter type. As an alternative, you can include the delimiter twice. Some examples are:

- "This is a string constant."
- ’So is this.’
- ’She said, "This is a string constant."’
- "She said, ""This is a string constant."""

Use the following code to include a literal quotation mark as part of your string:

```
&cDblQuote = ’"’; /* singlequote doublequote singlequote */
```

The following also produces a string with two double quotation marks in it:

```
&cDblQuote = """"; /* dquote dquote dquote dquote */
```

You can also directly embed the doubled double quotation mark in strings, such as:

```
&sImage = Char(10) | ’<IMG SRC="%IMAGE(’ | &pImageName | ’)"’;
```

Boolean Constants

Boolean constants represent a truth value. The two possible values are True and False.

Null Constant

Null constants represent an object reference value that does not refer to a valid object. This means that calling a method on the object or trying to get or set a property of it fails. The Null constant is just the keyword Null.

User-Defined Constants

You can define constants at the start of a PeopleCode program. Then you can use the declared constant anywhere that the corresponding value would be allowed. Constants can be defined as numbers, strings, or Boolean values.

User-defined constants can only be declared as Local.

The following is an example of user-defined constant declarations:

```
Constant &Start_New_Instance = True;
Constant &Display_Mode = 0;
Constant &AddMode = "A":
```

```
Local Field &Start_Date;
. . .
MyFunction(&Start_New_Instance, &Display_Mode, &Add_Mode);
```

See Also

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

Functions as Expressions

You can use any function that returns a value as an expression. The function can be used on the right side of an assignment statement, passed as a parameter to another function, or combined with other expressions to form a compound expression.

See Also

Chapter 2, "Understanding the PeopleCode Language," Functions, page 14

System Variables

System variables are preceded by a percent (%) symbol whenever they appear in a program. Use these variables to get the current date and time, or to get information about the user, the current language, the current record, page, or component, and more.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "System Variables"

Metastrings

Metastrings are special SQL expressions. The metastrings, also called meta-SQL, are preceded with a percent (%) symbol, and can be included directly in string literals. They expand at runtime into an appropriate substring for the current database platform. Metastrings are used in the following:

- SQLExec
- Scroll buffer functions (ScrollSelect and its relatives).
- In PeopleSoft Application Designer to construct dynamic views.
- With some rowset object methods (Select, SelectNew, Fill, and so on).
- With SQL objects.
- In PeopleSoft Application Engine.
- With some record class methods (Insert, Update, and so on).
- With COBOL.

See Also

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

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

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

Record Field References

Use record field references to retrieve the value stored in a record field, or to assign a value to a record field.

Record Field Reference Syntax

References to record fields have the following form:

```
[recordname.]fieldname
```

You must supply the *recordname* only if the record field and your PeopleCode program are in different record definitions.

For example, suppose that in a database for veterinarians you have two records, PET_OWNER and PET. A program in the record definition PET_OWNER must refer to the PET_BREED record field in the PET record definition as PET.PET_BREED.

However, a program in the PET record definition can refer to this same record field more directly as PET_BREED.

If the program is in the PET_BREED record field itself, it can refer to this record field using the caret (^) symbol.

The PeopleCode Editor replaces the caret symbol with the actual record field name.

You can also use object dot notation to refer to record fields, for example:

```
&FIELD = GetRecord(RECORD.PET_OWNER).GetField(FIELD.PET_BREED);
```

See Chapter 4, “Referencing Data in the Component Buffer,” page 41.

Legal Record Field Names

A record field name consists of two parts, the record name and the field name, separated by a period.

The field names used in PeopleCode are consistent with the field names allowed in the field definition. Case is ignored, although the PeopleCode Editor for the sake of convention, automatically formats field names in uppercase. A field name can be 1 to 18 characters, consisting of alphanumeric characters determined by your current language setting in Microsoft Windows, and characters #, @, $, and _.

A record name can be 1 to 15 characters, consisting of alphanumeric letters determined by your current language setting in Microsoft Windows, and characters #, @, $, and _.

Definition Name References

Definition name references are special expressions that reference the name of a PeopleTools definition, such as a record, page, component, business interlink, and so on. Syntactically, a definition name reference consists of a reserved word indicating the type of definition, followed by a period, then the name of the PeopleTools definition. For example, the definition name reference RECORD.BUS_EXPENSE_PER refers to the definition name BUS_EXPENSE_PER.

Generally, definition name references are passed as parameters to functions. If you attempt to pass a string literal instead of a definition name reference to such a function, you receive a syntax error.

You also use definition name references outside function parameter lists, for example in comparisons:

```
If (%Page = PAGE.SOMEPAGE) Then
   /* do stuff specific to SOMEPAGE */
End-If;
```

In these cases, the definition name reference evaluates to a string literal. Using the definition name reference instead of a string literal enables PeopleTools to maintain the code if the definition name changes.

If you use the definition name reference, and the name of the definition changes, the change automatically ripples through the code, so you do not have to change it or maintain it.

In the PeopleCode Editor, if you place your cursor over any definition name reference and right-click, you can select View Definition to open the definition.

In addition, for most definitions, if you specify a definition that hasn't been created in PeopleSoft Application Designer, you receive an error message when you try to save your program.

Legal and Illegal Definition Names

Legal definition names, as far as definition name references are concerned, consist of alphanumeric letters determined by your current language setting in Microsoft Windows, and the characters #, @, $, and _.

In some cases, however, the definition supports the use of other characters. You can, for example, have a menu item named A&M stored in the menu definition even though & is an illegal character in the definition name reference. The illegal character results in an error when you validate the syntax or attempt to save the PeopleCode.

You can avoid this problem in two ways:

- Rename the definition so that it uses only legal characters.
- Enclose the name of the definition in quotation marks in the reference, for example: ITEMNAME."A&M"

The second solution is a commonly used workaround in cases where the definition name contains illegal characters. If you use this notation, the definition name reference is not treated as a string literal: PeopleTools maintains the reference the same way as it does other definition name references.

Note. If your definition name begins with a number, you must enclose the name in quotation marks when you use it in a definition name reference. For example, CompIntfc."1_DISCPLIN_ACTN".

Reserved Word Summary Table

The following table summarizes the reserved words used in definition name references.

Reserved Word	Common Usage
BARNAME	Used with transfers and modal transfers.
BUSACTIVITY	Used with TriggerBusinessEvent.
BUSEVENT	Used with TriggerBusinessEvent.
BUSPROCESS	Used with TriggerBusinessEvent.
COMPINTFC	Used with Component Interface Classes.

Reserved Word	Common Usage
COMPONENT	Used with transfers and modal transfers, as well as for generating URLs.
FIELD	Used with methods and functions to designate a field.
FILELAYOUT	Used with the SetFileLayout File class method.
HTML	Used with the GetHTMLText function.
IMAGE	Used in with functions and methods to designate an image.
INTERLINK	Used with the GetInterlink function.
ITEMNAME	Used with transfers and modal transfers.
MENUNAME	Used with transfers and modal transfers.
MESSAGE	Used with Messaging functions and methods.
MOBILEPAGE	Used to identify a mobile page (used with transfers.)
NODE	Used with transfers and modal transfers, as well as generating URLs.
PAGE	Used with transfers and modal transfers to pass the page item name (instead of the page name), and with controls and other functions to pass the page name.
PORTAL	Used with transfers and modal transfers, as well as generating URLs.
RECORD	Used in functions and methods to designate a record.
SCROLL	The name of the scroll area in the page. This name is always equal to the primary record of the scroll.
SQL	Used with SQL definitions.
STYLESHEET	Used with style sheets.

See Also

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

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

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

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Component Interface Classes"

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

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

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

Variables

This section discusses.

- Supported variable types.
- User-defined variables.
- User-defined variable declaration and scope.
- Variable declaration.
- User-defined variable initialization.
- Scope of local variables.
- Lifetime of local variables.
- Restrictions on variable use.
- Variables and functions.
- Recursive functions.
- State of shared objects using PeopleSoft Pure Internet Architecture.

See Also

Chapter 2, "Understanding the PeopleCode Language," System Variables, page 19

Supported Variable Types

PeopleCode supports these types of variables:

User-defined variables These variable names are preceded by an & character wherever they appear in a program. Variable names can be 1 to 1000 characters, consisting of letters A-Z and a-z, digits 0-9, and characters #, @, $, and _.

System variables System variables provide access to system information. System variables have a prefix of the % character, rather than the & character. Use these

variables wherever you can use a constant, passing them as parameters to functions or assigning their values to fields or to temporary variables.

User-Defined Variables

A user-defined variable can hold the contents of a record field for program code clarity. For example, you may give a variable a more descriptive name than a record field, based on the context of the program. If the record field is from another record, you may assign it to a temporary variable rather than always using the record field reference. This makes it easier to enter the program, and can also make the program easier to read.

Also, if you find yourself calling the same function repeatedly to get a value, you may be able to avoid some processing by calling the function once and placing the result in a variable.

User-Defined Variable Declaration and Scope

The difference between the variable declarations concerns their life spans:

- Global

  The variable is valid for the entire session.

- Component

  The variable is valid while any page in the component in which the variable is defined stays active.

- Local

  The variable is valid for the duration of the PeopleCode program or function in which the variable is defined.

You can declare variables using the Global, Local, or Component statements, or you can use local variables without declaring them. Here are some examples:

```
Local Number &AGE;
Global String &OPER_NICKNAME;
Component Rowset &MY_ROWSET;
Local Any &SOME_FIELD;
Local ApiObject &MYTREE;
Local Boolean &Compare = True;
```

Variable declarations are usually placed above the main body of a PeopleCode program (along with function declarations and definitions). The exception is the Local declaration, which you can use within a function or the main section of a program. You can declare variables as any of the PeopleCode data types. If a variable is declared as an Any data type, or if a variable is not declared, PeopleTools uses an appropriate data type based on context.

Note. Declare a variable as an explicit data type unless the variable will hold a value of an unknown data type.

Global variables can be accessed from different components and applications, including an Application Engine program. A global variable must be declared, however, in each PeopleCode program where it's used. Use global variables rarely, because they are difficult to maintain.

Global variables are not available to a portal or applications on separate databases.

Component variables remain defined and keep their values while any page in the component in which they're defined remains active. Similar to a global variable, a component variable must be declared in each PeopleCode program where it's used.

Component variables act the same as global variables when an Application Engine program is called from a page (using CallAppEngine).

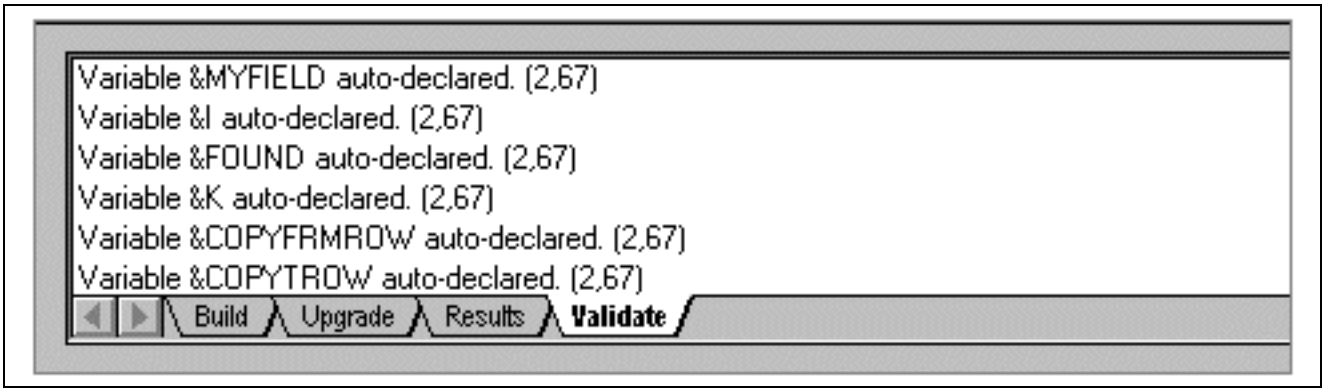
Component variables remain defined after a TransferPage, DoModal, or DoModalComponent function. However, variables declared as Component do not remain defined after using the Transfer function, whether you're transferring within the same component or not.

Local variables declared at the top of a PeopleCode program (or within the main, that is, non-function, part of a program) remain in scope for the life of that PeopleCode program. Local variables declared within a function are valid to the end of the function and not beyond.

You can check the values of Local, Global, and Component variables at runtime in the different variable windows of the PeopleCode debugger. Local variables declared within a function are displayed in the Function Parameters window.

Declaring Variables

Declare variables before you use them. If you do not declare a variable, it's automatically declared with the scope Local and the data type Any. You receive a warning message in the Validation tab of the PeopleSoft Application Designer output window for every variable that is not declared when you save the PeopleCode program, as shown in the following example:



Validation tab with auto-declared variables

If you've declared all the variables, you can use these values to ensure you do not have misspellings. For example, if you declared a variable as &END_DATE, then accidentally spell it as &EDN_DATE, the "new variable" appears on the Validate tab when you save the program.

Another reason to declare variables is for the design-time checking. If you declare a variable of one data type, then assign to it a value of a different type, the PeopleCode Editor catches that assignment as a design-time error when you try to save the program. With an undeclared variable, the assignment error does not appear until runtime.

The following example produces a design-time error when you try to save the program:

```
Local Field &DATE;

&DATE = GetRecord(RECORD.DERIVED_HR);
```

In addition, if you declare variables, the Find Object Reference feature finds embedded definitions. For example, suppose you wanted to find all occurrences of the field DEPT_ID. If you have not declared &MyRecord as a record, Find Object References does not find the following reference of the field DEPT_ID:

```
&MyRecord.DEPT_ID.Visible = False;
```

User-Defined Variable Initialization

To declare and initialize variables in one step, use the following format:

```
Local String &MyString = "New";
Local Date &MyDate = %Date;
```

This is available only for variables with the scope of Local.

Though you can declare more than one variable on a single line, you can only initialize one variable on a line. The following code creates a syntax error when you try to save the program:

```
Local Number &N1, &N2 = 5;
```

You cannot declare a variable, then initialize it in a second declaration statement. The following produces a duplicate declaration error when you try to save the program:

```
Global Number &N1;
...
Local String &N1 = "Str";  /* Duplicate definition. */
```

If you do not initialize variables, either when you declare them or before you use them, strings are initialized as Null strings, dates and times as Null, and numbers as zero.

Restrictions on Variable Use

The following data types can only be declared as Local:

- JavaObject
- Interlink

Note. Interlink objects can be declared as type Global in an Application Engine program.

- TransformData
- XmlNode

The following ApiObject data type objects can be declared as Global:

- Session
- PSMessages collection
- PSMessage
- All tree classes (trees, tree structures, nodes, levels, and so on.)
- Query classes

All other ApiObject data type objects must be declared as Local.

Scope of Local Variables

There are two types of local variables: program-local and function-local.

- A program-local variable is declared as local in the main part of the program, and is local to that program.
- A function-local variable is declared as local inside a function, and are only local to that function.

See Chapter 2, “Understanding the PeopleCode Language,” Recursive Functions, page 28.

A program-local variable can be affected by statements anywhere in the program. For example, suppose there are two functions in RECORD_A.FIELD_A.FieldFormula, FUNC_1 and FUNC_2, and both modify a local variable named &TEMP. They could affect each other, as they're both using the same variable name in the same PeopleCode program.

If, however, FUNC_3 is defined in RECORD_B_FIELD_B.FieldFormula and makes reference to &TEMP, it is *not* the same &TEMP as in RECORD_A.FIELD_A.FieldFormula. This becomes important when FUNC_1 calls FUNC_3. Technically both functions exist at the same time, one inside the other, but &TEMP is a different variable for each of them. However if FUNC_1 calls FUNC_2, then &TEMP is the same variable for both.

Duration of Local Variables

A local variable is valid for the duration of the PeopleCode program or function in which it's defined. A PeopleCode program is defined as what the PeopleCode Editor in Application Designer presents in a single window: a chunk of PeopleCode text associated with a single item (a record field event, a component record event, and so on.)

When the system evaluates a PeopleCode program and calls a function in the same PeopleCode program, a new program evaluation is not started.

However, when a function from a different PeopleCode program is called (that is, some PeopleCode text associated with a different item), the current PeopleCode program is suspended, and the Component Processor starts evaluating the new program. This means that any local variables in the calling program (called A) are no longer available. Those in the called program (called B) are available.

Even if the local variables in the A program have the same name as those in the B program, they are different variables and are stored separately.

If the called program (B) in turn calls a function in program A, a new set of program A's variables are allocated, and the called function in A uses these new variables. Thus, this second use of program A gets another lifetime, until execution returns to program B.

The following is an example of pseudocode to show how this might work. (This is non-compiled, non-working code. To use this example, you'd have to enter a similar program without the external declaration of the function in the other, not yet compiled, one.)

```
Program A (Rec.Field.FieldChange):
local number &temp;
declare function B1 PeopleCode Rec.Field FieldFormula;
/* Uncomment this declaration and comment above to compile this the first time.
   function B1
   end-function;
*/

function A1
WinMessage("A1: &temp is " | &temp);
&temp = &temp + 1;
A2();
B1();
A2();
end-function;

function A2
WinMessage("A2: &temp is " | &temp);
```

```
&temp = &temp + 1;
end-function;

A1();

Program B (Rec.Field.FieldFormula):
local number &temp;
declare function A2 PeopleCode Rec.Field FieldChange;

function B1
WinMessage("B1: &temp is " | &temp);
&temp = &temp + 1;
A2();
end-function;
```

When this is compiled and run, it produces the following output:

```
A1: &temp is 0
A2: &temp is 1
B1: &temp is 0
A2: &temp is 0
A2: &temp is 2
```

Variables and Functions

PeopleCode variables are always passed to functions by reference. This means, among other things, that a function can change the value of a variable passed to it so that the variable has the new value on return to the calling routine.

For example, the Amortize built-in function expects you to pass it variables into which it places the amount of a loan payment applied towards interest (&PYMNT_INTRST), the amount of the payment applied towards principal (&PYMNT_PRIN), and the remaining balance (&BAL). It calculates these values based on information that the calling routine supplies in other parameters:

```
&INTRST_RT=12;
&PRSNT_BAL=100;
&PYMNT_AMNT=50;
&PYMNT_NBR=1;
Amortize(&INTRST_RT, &PRSNT_BAL, &PYMNT_AMNT, &PYMNT_NBR,
&PYMNT_INTRST, &PYMNT_PRIN, &BAL);
&RESULT = "Int=" | String(&PYMNT_INTRST) | " Prin=" |
String(&PYMNT_PRIN) | "  Bal=" | String(&BAL);
```

Recursive Functions

PeopleCode supports true recursive functions. A function can call itself, and each possibly recursive call of the function has its own independent copy of the parameters and function-local variables.

When writing recursive functions, be careful about passing variables as parameters, because PeopleCode implements such calls by reference. This means that if you call a function such as:

```
Function Func(&n as Number)
&n = 3;
End-Function;
local &x = 5;
Func(&x);
```

After the call to Func(&x), &x has the value 3, not 5. If the call was Func(Value(&x)), after the call &x is still 5.

State of Shared Objects Using PeopleSoft Pure Internet Architecture

Consider the following scenario:

- A local and a global variable refer to the same object.
- That object is used in a modal component.
- Instead of completing the modal component, the user clicks the browser's Back button.

In general, the global state of the object is restored. If the object hasn't been destroyed from the global state, the global state of the object is used for local references; otherwise, the local state is used for local references.

Here's an example:

```
Global array of number &Global_Array;
Local array of number &Local_Array:

&Global_Array = CreateArray(1, 2, 3);
&Local_Array = &Global_Array
DoModal(Page.PAGENAME, "", -1, -1, 1, Record.SHAREDREC, 1);
/* return to here */
&Local_Array[1] = -1;
&Global_Array[2] = -2;
WinMessage(&Local_Array is " | &Local_Array.Join());
WinMessage(&Global_Array is " | &Global_Array.Join());
```

The following program, program 2, is located on the modal page the user is transferred to:

```
Global array of number &Global_Array;
&Global_Array[3] = -3;
```

The following program, program 3, is also located on the modal page:

```
Global array of number &Global_Array;
&Global_Array = CreateArray(1, 2, -3);
```

If program 2 is run, the output is the following:

&Local_Array is -1, -2, -3

&Global_Array is -1, -2, -3

However, if program 3 is run, thereby destroying the original global state, the output is the following:

&Local_Array is -1, 2, 3

&Global_Array is 1, -2, -3

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "System Variables"

Chapter 16, "Debugging Your Application," page 251

Operators

PeopleCode expressions can be modified and combined using math, string, comparison, and Boolean operators.

This section discusses:

- Math operators.
- Operations on dates and times.
- String concatenation.
- @ operator.
- Comparison operators.
- Boolean operators.

Math Operators

PeopleCode uses standard mathematical operators:

- +

  Add.

- -

  Subtract (or unary negative sign).

- *

  Multiply.

- /

  Divide.

- **

  Exponential.

Exponentiation occurs before multiplication and division; multiplication and division occur before addition and subtraction. Math expressions otherwise are evaluated from left to right. You can use parentheses to force the order of operator precedence.

The minus sign can also, of course, be used as a negation operator, as in the following expressions:

```
-10
- &NUM
```

```
- Product(&PERCENT_CUT, .01, SALARY)
```

Operations on Dates and Times

You can add or subtract two date values or two time values, which provides an Number result. In the case of dates, the number represents the difference between the two dates in days. In the case of time, the number represents the difference in seconds. You can also add and subtract numbers to or from a time or date, which results in another date or time. Again, in the case of days, the number represents days, and in the case of time, the number represents seconds.

The following table summarizes these operations:

Operation	Result Type	Result Represents
Time + number of seconds	Time	Resulting time
Date + number of days	Date	Resulting date
Date - date	Number	Difference in days
Time - time	Number	Difference in seconds
Date + time	DateTime	Date and time combined

String Concatenation

The string concatenation operator (|) is used to combine strings. For example, assuming &OPER_NICKNAME is “Dave”, the following statement sets &RETORT to “I can’t do that, Dave.”

```
&RETORT = "I can’t do that, " | &OPER_NICKNAME |  "."
```

The concatenation operator automatically converts its operands to strings. This makes it easy to write statements that display mixed data types. For example:

```
&DAYS_LEFT = &CHRISTMAS - %Date;
WinMessage("Today is " | %Date | ".  Only " | &DAYS_LEFT | " shopping days left!");
```

@ Operator

The @ operator converts a string storing a definition reference into the definition. This is useful, for example, if you want to store definition references in the database as strings and retrieve them for use in PeopleCode; or if you want to obtain a definition reference in the form of a string from the operator using the Prompt function.

To take a simple example, if the record field EMPLID is currently equal to 8001, the following expression evaluates to 8001:

```
@"EMPLID"
```

The following example uses the @ operator to convert strings storing a record reference and a record field reference:

```
&STR1 = "RECORD.BUS_EXPENSE_PER";
&STR2 = "BUS_EXPENSE_DTL.EMPLID";
&STR3 = FetchValue(@(&STR1), CurrentRowNumber(1), @(&STR2), 1);
WinMessage(&STR3, 64);
```

Note. String literals that reference definitions are not maintained by PeopleTools. If you store definition references as strings, then convert them with the @ operator in the code, this creates maintenance problems whenever definition names change.

The following function takes a rowset and a record, passed in from another program, and performs some processing. The GetRecord method does not take a variable for the record, however, you can dereference the record name using the @ symbol. Because the record name is never hard-coded as a string, if the record name changes, this code does not have to change.

```
Function Get_My_Row(&PASSED_ROWSET, &PASSED_RECORD)

   For &ROWSET_ROW = 1 To &PASSED_ROWSET.RowCount
      &UNDERLYINGREC = "RECORD." | &PASSED_ROWSET.DBRecordName;
      &ROW_RECORD = &PASSED_ROWSET.GetRow(&ROWSET_ROW).GetRecord(@&UNDERLYINGREC);

      /* Do other processing */

   End-For;

End-Function;
```

Comparison Operators

Comparison operators compare two expressions of the same data type. The result of the comparison is a Boolean value. The following table summarizes these operators:

Operator	Meaning
=	Equal
!=	Not equal
<>	Not equal
<	Less than
<=	Less than or equal to

Operator	Meaning
>	Greater than
>=	Greater than or equal to

You can precede any of the comparison operators with NOT, for example:

- Not=
- Not<
- Not>=

Expressions formed with comparison operators form logical terms that can be combined using Boolean operators.

String comparisons are case-sensitive. You can use the Upper or Lower built-in functions to do a case-insensitive comparison.

See Also

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

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

Boolean Operators

The logical operators AND, OR, and NOT are used to combine Boolean expressions. The following table shows the results of combining two Boolean expressions with AND and OR operators:

Expression 1	Operator	Expression 2	Result
FALSE	AND	FALSE	FALSE
FALSE	AND	TRUE	FALSE
TRUE	AND	TRUE	TRUE
FALSE	OR	FALSE	FALSE
FALSE	OR	TRUE	TRUE
TRUE	OR	TRUE	TRUE

The NOT operator negates Boolean expressions, changing a True value to False and a False value to True.

In complex logical expressions using the operations AND, OR, and NOT, NOT takes the highest precedence, AND is next, and OR is lowest. Use parentheses to override precedence. (It's generally a good idea to use parentheses in logical expressions anyway, because it makes them easier to decipher.) If used on the right side of an assignment statement, Boolean expressions must be enclosed in parentheses.

The following are examples of statements containing Boolean expressions:

```
&FLAG = (Not (&FLAG)); /* toggles a Boolean */
if ((&HAS_FLEAS or &HAS_TICKS) and
SOAP_QTY <= MIN_SOAP_QTY)  then
  SOAP_QTY = SOAP_QTY + OrderFleaSoap(SOAP_ORDER_QTY);
end-if;
```

CHAPTER 3

Understanding Objects and Classes in PeopleCode

This chapter provides an overview of classes and objects and discusses how to:

- Work with objects.
- Assign objects.
- Pass objects.

Understanding Classes and Objects

PeopleSoft delivers classes of objects that you can manipulate with PeopleCode. In addition, you can extend the existing classes or create your own. The delivered classes may or may not have a graphic user interface equivalent; some are representations of data structures that occur only at runtime. With PeopleCode, you can manipulate data in the data buffer easily and consistently. These classes enable you to write code that's more readable, more easily maintained, and more useful.

This section discusses:

- Classes.
- Objects.
- Object instantiation.

Classes

A *class* is the formal definition of an object and acts as a template from which an instance of an object is created at runtime. The class defines the properties of the object and the methods used to control the object's behavior.

PeopleSoft delivers predefined classes (such as Array, File, Field, SQL, and so on). You can create your own classes using the Application class. You can also extend the functionality of the existing classes using the Application class.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Application Classes"

Objects

An *object* represents a unique instance of a data structure defined by the template provided by its class. Each object has its own values for the variables belonging to its class and responds to methods defined by that class. This is the same for classes provided by PeopleSoft and for classes you create yourself.

After an object has been created (instantiated) from a class, you can change its properties. A *property* is an attribute of an object. Properties define:

- Object characteristics, such as name or value.
- The state of an object, such as deleted or changed.

Some properties are read-only and cannot be set, such as Name or Author. Other properties can be set, such as Value or Label.

Objects are different from other data structures. They include code (in the form of *methods*), not just static data. A *method* is a procedure or routine, associated with one or more classes, that acts on an object.

An analogy to illustrate the difference between an object and its class is the difference between a car and the blue Citroen with license plate number TS5800B. A class is a general category, while the object is a specific instance of that class. Each car comes with standard characteristics, such as four wheels, an engine, or brakes, that define the class and are the template from which the individual car is created. You can change the properties of an individual car by personalizing it with bumper stickers or racing stripes, which is like changing the Name or Visible property of an object. The model and date that the car is created are similar to read-only properties because you cannot alter them. A tune-up acts on the individual car and changes its behavior, much as a method acts on an object.

Object Instantiation

A class is the blueprint for something, like a bicycle, a car, or a data structure. An object is the actual thing that's built using that class (or blueprint.) From the blueprint for a bicycle, you can build a specific mountain bike with 23 gears and tight suspension. From the blueprint of a data structure class, you build a specific instance of that class. *Instantiation* is the term for building that copy, or an instance, of a class.

Creating and Using Objects

This section discusses how to:

- Instantiate objects.
- Change object properties.
- Invoke methods.

Instantiating Objects

Generally you instantiate objects (create them from their classes) using built-in functions or methods of other objects. Some objects are instantiated from data already existing in the data buffer. Think about this kind of object instantiation as taking a chunk of data from the buffer, encapsulating it in code (methods and properties), manipulating it, then freeing the references. Some objects can be instantiated from a previously created definition, such as a page or file layout definition, instead of from data.

The following example creates a field object:

```
Local field &MyField

&MyField = GetField();
```

Get functions, which include functions such as GetField, GetRecord, and so on, generally provide access to data that already exists, whether in the data buffers or from an existing definition.

Create functions, which include functions such as CreateObject, CreateArray, CreateRecord, generally create defined objects that do not yet exist in the data buffer. Create functions create only a buffer structure. They do not populate it with data. For example, the following function returns a record object for a record that already exists in the component buffer:

```
&REC = GetRecord();
```

The following example creates a standalone record. However, there is no data in &REC2. The specified record definition must be created previously, but the record does not have to exist in either the component or data buffer:

```
&REC2 = CreateRecord(EMP_CHKLST_ITM);
```

Objects with no built-in functions can only be instantiated from a session object (such as tree classes, component interfaces, and so on). For most of these classes, when you use a Get function, all you get is an identifier for the object. To fully instantiate the object, you must use an Open method.

See Also

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

Changing Properties

To set or get characteristics of an object, or to determine the state of an object, you must access its properties through *dot notation* syntax. Follow the reference to the object with a period, followed by the property, and assign it a value. The format is generally as follows:

```
Object.Property = Value
```

The following example hides the field &MYFIELD:

```
&MYFIELD.Visible = False
```

You can return information about an object by returning the value of one of its properties. In the following example, &X is a variable that is assigned the value found in the field &MYFIELD:

```
&X = &MYFIELD.Value
```

In the following example, a property is used as the test for a condition:

```
If &ROWSET.ActiveRowCount <> &I Then
```

Invoking Methods

You also use dot notation to execute methods. Follow the reference to the object with a period, then with the method name and any parameters the method takes. The format is generally:

```
Object.method();
```

You can string methods and property values together into one statement. The following example strings together the GetField method with the Name property:

```
If &REC_BASE.GetField(&R).Name = &REC_RELLANG.GetField(&J).Name Then
```

Some methods return a Boolean value: True if the method executes successfully; False if it does not. The following method compares all like-named fields of the current record object with the specified record. This method returns as True if all like-named fields have the same value:

```
If &MYRECORD.CompareFields(&OTHERRECORD) Then
```

Other methods return a reference to an object. The GetCurrEffRow method returns a row object:

```
&MYROW = &MYROWSET.GetCurrEffRow();
```

Some methods do not return anything. Each method's documentation indicates what it returns.

Many objects have default methods. Instead of entering the name of the method explicitly, you can use that method's parameters. Objects with default methods are *composite objects;* that is, they contain additional objects within them. The default method is generally the method used to get the lower-level object.

A good example of a composite object is a record object. Record definitions are composed of field definitions. The default method for a record object is GetField.

The following lines of code are equivalent:

```
&FIELD = &RECORD.GetField(FIELD.EMPLID);
&FIELD = &RECORD.EMPLID;
```

Note. If the field you're accessing has the same name as a record property (such as NAME) you cannot use the shortcut method for accessing the field. You must use the GetField method.

Another example of default methods concerns rowsets and rows. Rowsets are made up of rows, so the default method for a rowset is GetRow. The two specified lines of code are equivalent: They both get the fifth row of the rowset:

```
&ROWSET = GetRowSet();

/*the next two lines of code are equivalent */

&ROW = &ROWSET.GetRow(5);
&ROW = &ROWSET(5);
```

The following example illustrates the long way of enabling the Name field on a second-level scroll area (the code is executing on the first-level scroll area):

```
GetRowset(SCROLL.EMPLOYEE_CHECKLIST).GetRow(1).
GetRecord(EMPL_CHKLST_ITM).GetField(FIELD.NAME).Enabled = True;
```

Using default methods enables you to shorten the previous code to the following:

```
GetRowset(SCROLL.EMPLOYEE_CHECKLIST)(1).EMPL_CHKLST_ITM.NAME.
Enabled = True;
```

Expressions of the form *class.name.property* or *class.name.method(..)* are converted to a corresponding object. For example, the code `&temp = RECORD.JOB.IsChanged;` is evaluated as if it were `&temp = GetRecord(RECORD.JOB).IsChanged;`.

Furthermore, the code `JOB.EMPLID.Visible = False;` is evaluated as if it were `GetField(JOB.EMPLID).Visible = False;`.

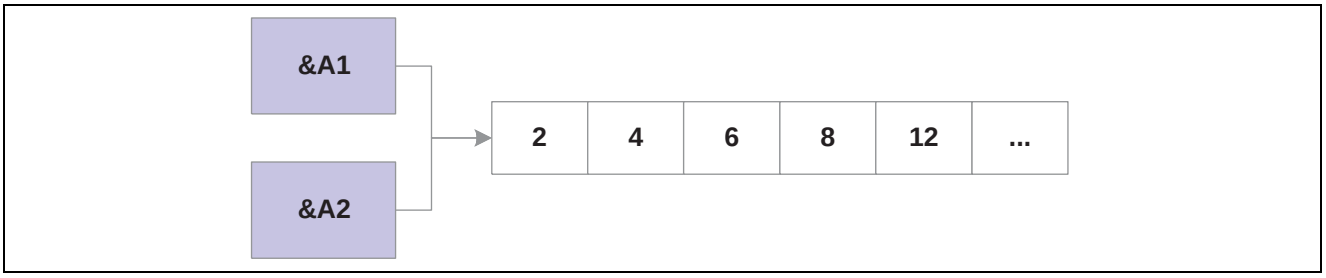
Assigning Objects

When you assign one object to another, you do not create a copy of the object, but only make a copy of the reference.

In the following example, &A1 and &A2 refer to the same object. The assignment of &A1 to &A2 does not allocate any database memory or copy any part of the original object. It makes &A2 refer to the same object to which &A1 refers.

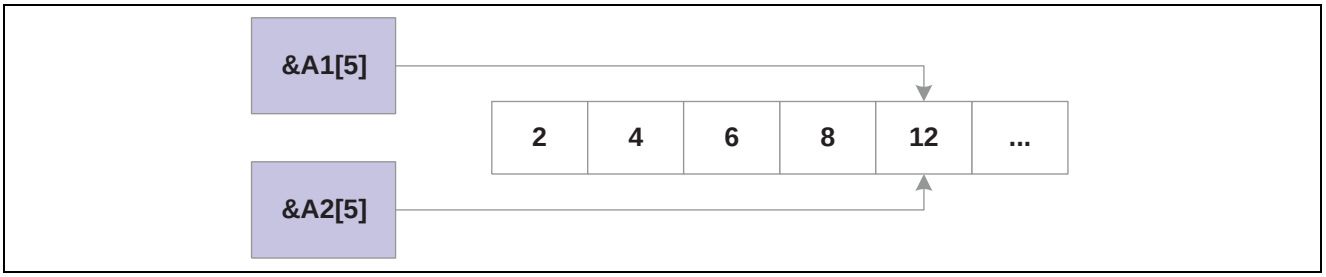
```
Local Array of Number &A1, &A2;

&A1 = CreateArray(2, 4, 6, 8, 10);
&A2 = &A1;
```

The following diagram shows how both references point to the same object:



Representation of two arrays

If the next statement is `&A2[5] = 12;`, then &A1[5] also equals 12, as shown in the following diagram:



Representation of two arrays with same content

The following example is *not* considered an object assignment:

```
Local number &NUM;
Local Array of Number &A1;

&A1 = CreateArray(2, 4, 6, 8, 10);
&NUM = &A1[3];
```

&NUM is of data type Number, which is not an object type. If you later change the value of &NUM in the program, you won't change the element in the array.

Passing Objects

All PeopleCode objects can be passed as function parameters. You can pass complex data structures between PeopleCode functions (as opposed to passing long lists of fields). If a function is passed an object, the function works on the actual object, not on a copy of the object.

In the following simple example, a reference to the Visible property is passed, not the *value* of Visible. This enables the MyPeopleCodeFunction either to get or set the value of the Visible property:

```
MyPeopleCodeFunction(&MyField.Visible);
```

In the following example, the function Process_Rowset loops through every row and record in the rowset it is passed and executes an Update statement on each record in the rowset. This function can be called from any PeopleCode program and can process any rowset that is passed to it.

```
Local Rowset &RS;
Local Record &REC;

Function Process_RowSet(&ROWSET as Rowset);

   For &I = 1 To &ROWSET.Rowcount
      For &J = 1 To &ROWSET.Recordcount
         &REC = &ROWSET.GetRow(&I).GetRecord(&J);
         &REC.Update();
      End-For;
   End-For;
End-Function;

&RS = GetLevel0();

Process_RowSet(&RS);
```

The following function takes a rowset and a record passed in from another program. GetRecord does not take a variable for the record; however, you can use the @ symbol to dereference the record name.

```
Function Get_My_Row(&PASSED_ROWSET, &PASSED_RECORD)

   For &ROWSET_ROW = 1 To &PASSED_ROWSET.RowCount
      &UNDERLYINGREC = "RECORD." | &PASSED_ROWSET.DBRecordName;
      &ROW_RECORD = &PASSED_ROWSET.GetRow(&ROWSET_ROW).GetRecord(@&UNDERLYINGREC);

      /* Do other processing */

   End-For;

End-Function;
```

CHAPTER 4

Referencing Data in the Component Buffer

This chapter provides an overview of component buffer structure and contents and discusses how to:

- Specify data with contextual references.
- Specify data with references using scroll path syntax and dot notation.

Understanding Component Buffer Structure and Contents

This section discusses:

- Component buffer contents.
- Rowsets and scroll areas.
- Record fields in the component buffer.

See Also

Chapter 4, "Referencing Data in the Component Buffer," Specifying Data with References Using Scroll Path Syntax and Dot Notation, page 49

Component Buffer Contents

PeopleCode frequently must refer to data in the component buffer, the area in memory that stores data for the currently active component.

There are two methods for specifying a piece of data in the component buffer from within PeopleCode:

- Contextual references, which refer to data relative to the location of the currently executing PeopleCode program.
- References using scroll path syntax, which provide a complete, or absolute, path through the component buffer to the referenced component.

In addition to the built-in functions used to access the component buffer, PeopleCode provides enhanced access to structured data buffers using the object syntax. Use the object-oriented PeopleCode to resolve contextual ambiguities when you reference a nonprimary record field that appears on more than one scroll level in a component. As with built-in functions, object syntax provides for both relative and absolute references to component buffer data.

See Chapter 3, "Understanding Objects and Classes in PeopleCode," page 35.

The component buffer consists of rows of buffer fields that hold data for the records associated with page controls, including primary scroll records, related display records, derived/work records, and Translate table records. PeopleCode can reference buffer fields associated with page controls and other buffer fields from the primary scroll record and related display records.

See Chapter 4, "Referencing Data in the Component Buffer," Record Fields and the Component Buffer, page 43.

Primary scroll records are the principal Structured Query Language (SQL) tables or views associated with page scroll levels. A primary scroll record uniquely identifies a scroll level in the context of its page: each scroll level can have only one primary scroll record, and the same primary scroll record cannot occur on more than one scroll area at the same level of the page. Parent-child relations between primary scroll records determine the dependency structure of the scroll areas on the page. The primary record on a level one scroll area must be a child of the primary record on level zero, the primary record on a level two scroll area must be a child of the primary record on its enclosing level one scroll area, and the primary record on a level three scroll area must be a child of the primary record on its enclosing level two scroll area.

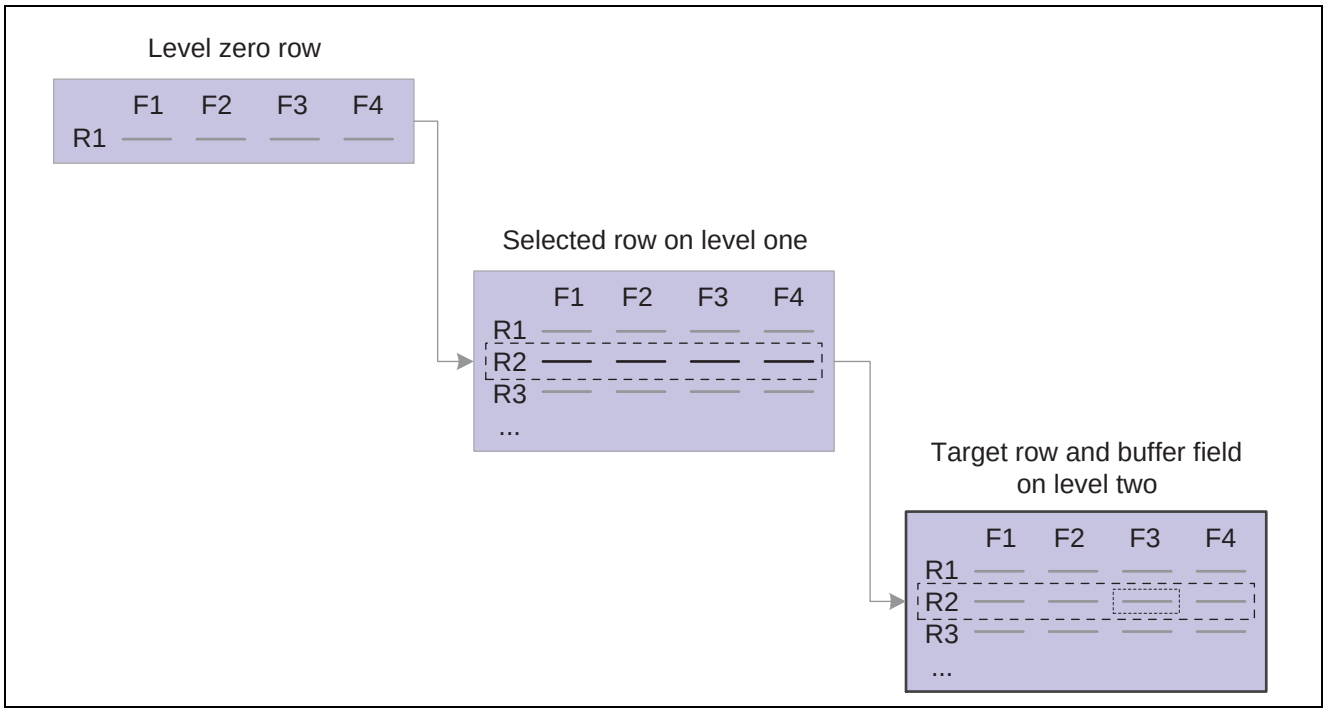
Note. Level zero may have multiple records.

The hierarchical relations among scroll areas, controlled by hierarchical relations among primary scroll records, enable the user and PeopleCode to drill down through the scroll hierarchy to access any buffer field, including related display, derived/work, and Translate table buffer fields, which occupy space on the same rows as the primary scroll record buffer fields with which they are associated.

For example, to access a page field on level two of a page, a user must:

1. Select a field on level one of the page.
2. Scroll to and select the field on level two of the page.

The following diagram illustrates this scroll path taken by the user:



Scroll path to a buffer field

To access the same field in the component buffer, PeopleCode must:

1. Specify a scroll area and row on scroll level one: this selects a subset of dependent rows on level two.
2. Specify a scroll area and row on scroll level two.
3. Specify the recordname.fieldname on the level two row.

PeopleCode component buffer functions use a common scroll path syntax for locating scrolls, rows, and fields in multiple-scroll pages.

Rowsets and Scroll Areas

Rowsets enable more consistent, more convenient, and less ambiguous manipulation of buffer data than previous built-in functions could achieve. It's a hierarchical data object that can represent an entire scroll area and all of its subordinate scroll areas.

A rowset can contain the entire contents of a component buffer, or the contents of any lower-level scroll area plus all of its subordinate buffer data. The hierarchical structure of component levels—scroll area, row, record, field—is provided by the new object data types, Rowset, Row, Record, and Field.

You can access any rowset, row, record, or field within the buffer using the dot notation inherent in PeopleTools 8 object-oriented programming. This enables you to reference fields within a record object, records within a row object, and rows within a rowset object as properties of the parent objects.

See Also

Chapter 5, "Accessing the Data Buffer," page 59

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

Chapter 3, "Understanding Objects and Classes in PeopleCode," page 35

Record Fields and the Component Buffer

The record fields in the component buffer are a superset of those accessible to the user through page controls. In most cases, PeopleCode can reference any record field in a scroll area's primary scroll record or in a related display record, not just those fields that are associated with page controls. The following table lists record types and locations:

Type and Location of Record	Presence in Component Buffer
Primary record on scroll levels greater than zero	On scroll levels greater than zero, all record fields from the primary scroll record are in the component buffer. PeopleCode can refer to any record field on the primary scroll record, even if it is not associated with a page control.

Type and Location of Record	Presence in Component Buffer
Primary record on scroll level zero	If scroll level zero of a page contains only controls associated with primary scroll record fields that are search keys or alternate search keys, then only the search key and alternate search key fieldsave in the component buffer, not the entire record. The values for the fields come from the keylist, and the record cannot run RowInit PeopleCode. If level zero contains at least one record field from the primary scroll record that is not a search key or alternate search key, then all the record fields from the primary scroll record are available in the buffer. (For this reason, you may sometimes need to add one such record field at level zero of the page to make sure that all the record fields of the level-zero primary record can be referenced from PeopleCode.)
Related display record fields	The buffer contains the related display record field, plus any record fields from the related display record that are referenced by PeopleCode programs. You can reference any record field in a related display record.
Derived/work record fields	Only derived/work record fields associated with page controls are in the component buffer. Other record fields from the derived/work record cannot be referenced from PeopleCode.
Translate table record fields	Only Translate table fields associated with page controls are available in the component buffer. Other fields from the Translate table cannot be referenced from PeopleCode.

Note. In RowSelect PeopleCode, you can refer only to record fields on the record that is currently being processed.

Specifying Data with Contextual References

In a *contextual reference*, PeopleCode refers to a row or buffer field determined by the context in which a PeopleCode program is currently executing.

This section includes an overview of the current context and discusses how to:

- Use contextual row references.
- Use contextual buffer field references.

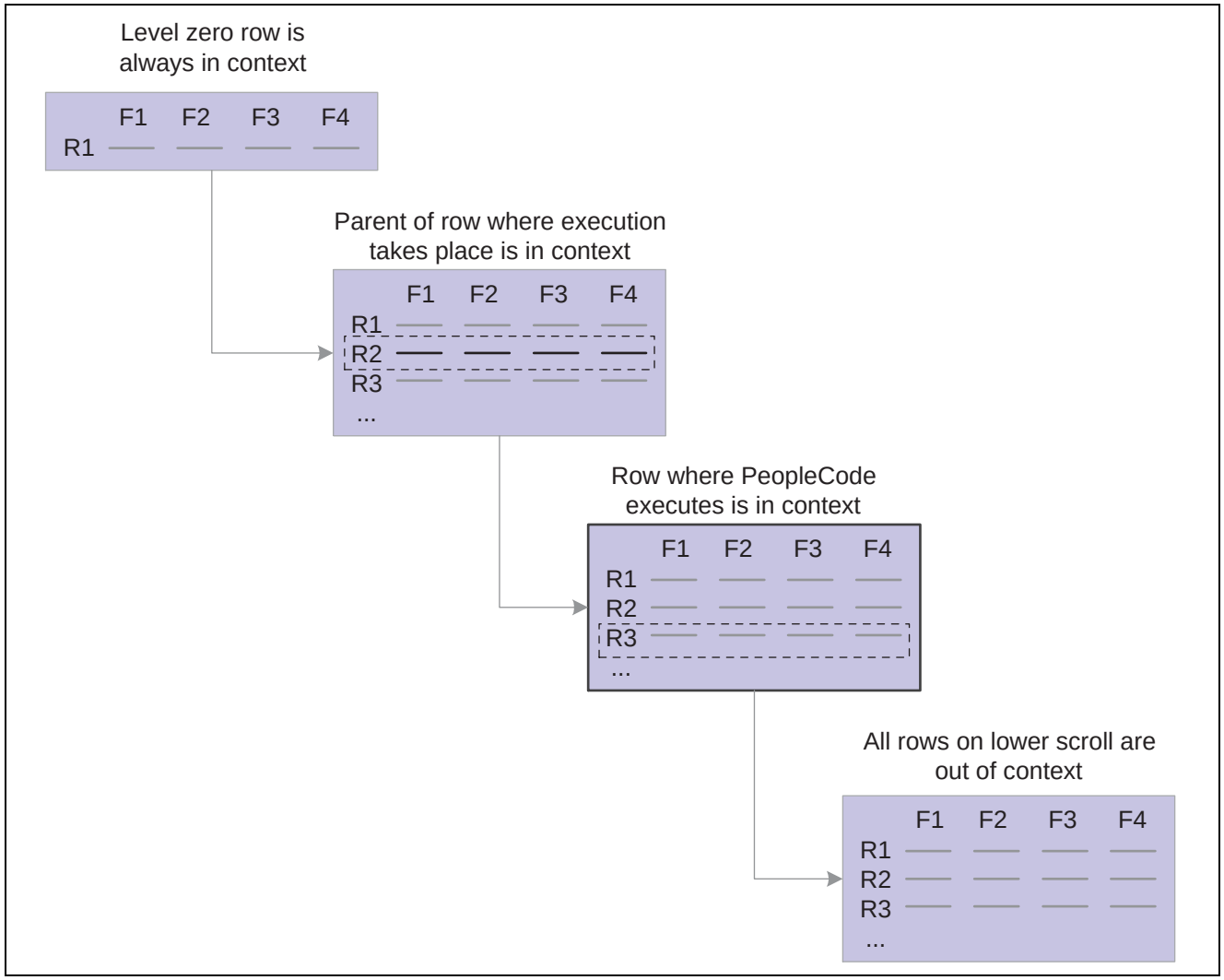
Understanding Current Context

All PeopleCode programs, with the exception of programs associated with standard menu items, execute in a current context. The current context determines which buffer fields can be contextually referenced from PeopleCode, and which row of data is the current row on each scroll level at the time a PeopleCode program is executing.

The current context comprises a subset of the buffer fields in the component buffer, determined by the row of data where a PeopleCode program is executing. The current context includes:

- All buffer fields in the row of data where the PeopleCode program is executing.
- All buffer fields in rows that are hierarchically superior to the row where the PeopleCode program is executing.

In the following diagram, all rows enclosed in dotted rectangles are part of the current context:



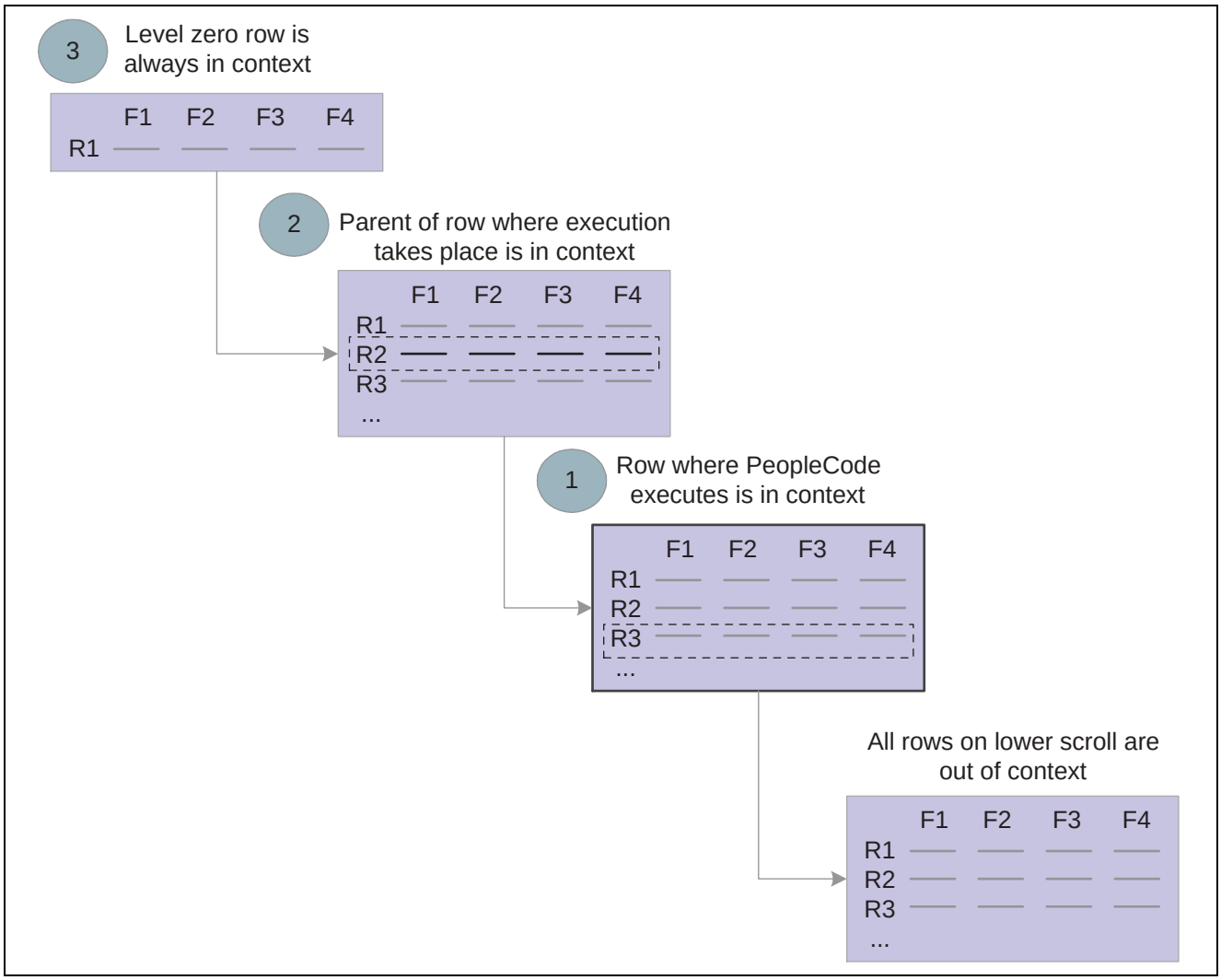
Context of PeopleCode executing on a level two scroll area

In the preceding diagram, a PeopleCode program is executing in a buffer field on row R3 on scroll level two. The rows in scroll level two are dependent on row R2 on scroll level one. The rows in scroll level one are dependent on the single row at scroll level zero. The current context consists of all the buffer fields at level two row R3, level one row R2, and level zero row R1. The rows in the current context on levels one and two are the current rows on their respective scroll areas. The single row on level zero is always current and is included in any current context. All rows other than the current rows and the level zero row are outside the current context. No current row can be determined on scroll areas below the one where the PeopleCode is executing.

With PeopleTools 8, contextual references work within the structure of a rowset object, and can include references to all field objects, record objects, row objects, and rowset objects in the current context.

Contextual Reference Processing Order

PeopleCode resolves contextual references at runtime by first checking the row where the PeopleCode program is executing. If PeopleCode does not find an appropriate buffer field, it looks in progressively higher rows in the current context. The following diagram indicates this processing order:



Processing order of a contextual reference

In typical pages, this processing order is not significant; however, if the same record occurs on more than one level of a page, it's important to understand how the direct reference is resolved.

Using Contextual Row References

A contextual row reference refers to a row in the current context on level one or lower in the page. Because each scroll area uses a unique primary record, the name of that record uniquely identifies whichever row is in the current context for that scroll level. A contextual row reference uses a **RECORD.***recordname* component name reference to specify the scroll level of the intended row, resulting in a reference to the current row at the specified scroll level.

For example, you can use contextual row references with the RecordDeleted, RecordNew, and RecordChanged functions:

```
If RecordDeleted(RECORD.SOME_REC) Then...
```

With PeopleTools 8 object-oriented programming, a row can be referenced by specifying parent rows or rowsets of the current rowset:

```
If GetRowSet().ParentRowset.ParentRow.IsDeleted Then...
```

In early versions of PeopleTools, you could make contextual row references using a *recordname.fieldname* expression:

```
HideRow(SOME_REC.ANY_FIELD)
If RecordDeleted(SOME_REC.ANY_FIELD) Then...
```

This syntax is still supported.

See Also

Chapter 4, “Referencing Data in the Component Buffer,” Understanding Current Context, page 45

Using Contextual Buffer Field References

A *contextual buffer field reference* is a type of PeopleCode expression that refers to a buffer field by specifying a record field. The row of the buffer field is determined by the current context of the PeopleCode program where the reference is made. You can use a contextual buffer field reference to retrieve or update the value in the buffer field, to pass the buffer field value to a function, or to reference an instance of a page control associated with the buffer field. The following statements use contextual buffer field references:

```
/* Assigns value of variable to buffer field */
SOME_RECORD.SOME_FIELD = &VAL;
/* Assigns value of buffer field to variable */
&VAL = SOME_RECORD.SOME_FIELD;
/* Hides instance of control associated with buffer field */
Hide(SOME_RECORD.SOME_FIELD);
```

With PeopleTools 8 object-oriented programming, a field object incorporates information about both the record field on which the buffer field is based and the page control with which the buffer field is associated. By referring to the field object, you either make a contextual buffer field reference or you change an interface attribute of the associated page control, depending on the object property you use. The following example has the same effect as a contextual buffer field reference:

```
/* Assigns value of a variable to a buffer field */
&MYFIELD.Value = &SOMEVAL;
```

Contextual Buffer Field Reference Ambiguity

Nonprimary record fields may appear on more than one scroll level in a page. For example, a page may use a derived/work field DERIVED_JS.CALC_1 as a work field on level one and level two of the same page. This creates distinct DERIVED_JS.CALC_1 buffer fields for rows on both levels. Because of the order in which PeopleCode resolves contextual buffer field references, if the contextual reference `&VAL = DERIVED_JS.CALC_1;` executes in a PeopleCode program on a level-two row, the reference always retrieves the buffer field value on the current row on level two. PeopleCode on level two is unable to retrieve the value of the DERIVED_JS.CALC_1 on level one using a contextual reference.

To explicitly reference the DERIVED_JS.CALC_1 buffer field on level one, use a component buffer function with a scroll path:

```
&VAL = FetchValue(SCROLL.level1_scrollname, CurrentRowNumber(1), DERIVED_JS.CALC_⇒
1);
```

The CurrentRowNumber function returns the current row on level one, or the parent row of the level two row where the PeopleCode program is executing.

Ambiguous Contextual References to Buffer Fields on Level Zero

Level zero of a page contains only a single row of data, and the buffer fields in this row are always in the current context. For this reason you can almost always refer to a level zero buffer field using a contextual reference. However, referential ambiguity can make it impossible to reference a buffer field on level zero contextually. For example, a page may use a derived/work field DERIVED_JS.CALC_1 as a work field on level zero and level one of the same page. This creates distinct DERIVED_JS.CALC_1 buffer fields for rows on both levels. Because of the order in which PeopleCode resolves contextual field references, if the `&VAL = DERIVED_JS.CALC_1;` contextual reference executes in a PeopleCode program on a level-one row, it always retrieves the buffer field value on the current row on level one.

To explicitly reference the DERIVED_JS.CALC_1 buffer field on level zero, you must use a component buffer function with this syntax:

```
Function([recordname.]fieldname, rownum)
```

Here *rownum*, because it is on level zero, is always equal to one. In the previous example of the DERIVED_JS.CALC_1 field, you would use this statement:

```
&VAL = FetchValue(DERIVED_JS.CALC_1, 1);
```

Ambiguous References with Objects

With PeopleTools 8 object-oriented programming, even if two field objects that are in different rowsets contain buffer data that's based on the same underlying record field, references to those objects are inherently unique, because each is instantiated with respect to a specific point in the hierarchy of the buffer. Any manipulation of a field object's interface properties affects only the page control with which it's associated.

The following example instantiates a field object using the long form, to emphasize the hierarchical form of the data. It then hides the field's associated page control. Because this is a unique instance of the field, based on its hierarchy, hiding this field does not affect the visibility of any other page control associated with the same record field:

```
&MYFIELD = GetRowset(SCROLL.EMPL_CHECKLIST).GetRow(&I).
GetRecord(RECORD.EMPL_CHECKLIST).GetField(EMPL_CHECKLIST.EMPLID);
&MYFIELD.Visible = False;
/* the same code, using the "short" form */
&MYFIELD = GetRowset(SCROLL.EMPL_CHECKLIST).GetRow(&I).
EMPL_CHECKLIST.EMPLID;
```

Note. Any change in a field object's value affects both the underlying record field and the value of any other field object oriented on the same record field. This behavior is the same as the behavior of contextual buffer field references that alter the field value.

See Also

Chapter 4, "Referencing Data in the Component Buffer," Specifying Data with References Using Scroll Path Syntax and Dot Notation, page 49

Specifying Data with References Using Scroll Path Syntax and Dot Notation

This section provides an overview of scroll paths and discusses:

- Structure scroll path syntax in PeopleTools 7.5.
- Reference scroll levels, rows, and buffer fields.

See Also

Chapter 5, "Accessing the Data Buffer," page 59

Understanding Scroll Paths

A *scroll path* is a construction found in the parameter lists of many component buffer functions, which specifies a scroll level in the currently active page. Additional parameters are required to locate a row or a buffer field at the specified scroll level.

PeopleTools 7.5 scroll path syntax enables you to eliminate ambiguous references, which, although rare, do sometimes occur in complex components.

See Chapter 4, "Referencing Data in the Component Buffer," Using Contextual Buffer Field References, page 47.

PeopleTools 8 adds the convenience of object-oriented dot notation and default methods, which produce inherently nonambiguous references, to PeopleCode programs. There are examples of dot notation in this section and examples of the scroll path syntax available in PeopleTools 7.5, which is still valid in PeopleTools 8.

Structuring Scroll Path Syntax in PeopleTools 7.5

PeopleTools 7.5 offers two constructions for scroll paths: a standard scroll path syntax, which in all cases except data buffer references is identical to the syntax in PeopleTools 6 and 7; and an alternative syntax using a **SCROLL.***scrollname* expression. The latter is more powerful in that it can process some rare cases where a **RECORD.***recordname* expression results in an ambiguous reference.

Scroll Path Syntax with RECORD.recordname

Here is the standard scroll path syntax:

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

If the target level (the level you want to reference) is one, you must supply only the **RECORD.***target_recname* parameter. If the target scroll level is greater than one, you must provide scroll name and row level parameters for all hierarchically superior scroll levels, beginning at level one. The following table indicates the scroll path syntax for the three possible target scroll levels:

Target Level	Scroll Path Syntax
1	`RECORD.target_recname`
2	`RECORD.level1_recname, level1_row, RECORD.target_recname`
3	`RECORD.level1_recname, level1_row, RECORD.level2_⇒ recname, level2_row, RECORD.target_recname`

If you are referring to a row or a buffer field, additional parameters are required after the scroll path.

The following are the standard scroll path syntax parameters:

Syntax Parameters	Description
RECORD.*level1_recname*	Specifies the name of a record associated with scroll level one, normally the primary scroll record. This parameter is required if the target scroll level is two or three.
level1_row	An integer that selects a row on scroll level one. This parameter is required if the target scroll level is two or three.
RECORD.*level2_recname*	Specifies the name of a record associated with scroll level two, normally the primary scroll record. This parameter is required if the target row is on scroll level three.
level2_row	An integer that selects a row on scroll level two. This parameter is required if the target row is on scroll level three.
RECORD.*target_recname*	Specifies a record associated with the target scroll level, generally the primary scroll record. The scroll can be on level one, two, or three of the active page.

Scroll Path Syntax with SCROLL.scrollname

As an alternative to **RECORD**.*recordname* expressions in scroll path constructions, PeopleTools 7.5 permits use of a **SCROLL.***scrollname* expression. Scroll paths using **SCROLL.***scrollname* are functionally identical to those using **RECORD.***recordname*, except that **SCROLL**.*scrollname* expressions are more strict: they can refer only to a scroll level's primary record; whereas **RECORD.***recordname* expressions can refer to any record in the scroll level, which in some rare cases can result in ambiguous references. (This can occur, for example, if the **RECORD.***recordname* expression inadvertently references a related display record in another page in the component.) Use of **RECORD.***recordname* is still permitted, and there is no requirement to use the **SCROLL.***scrollname* alternative unless it is needed to avoid an ambiguous reference.

The *scrollname* is the same as the scroll level's primary record name.

The scroll name cannot be viewed or changed through the PeopleSoft Application Designer interface. Here is the complete scroll path syntax using **SCROLL.***scrollname* expressions:

```
[SCROLL.level1_scrollname, level1_row, [SCROLL.level2_scrollname, level2_row, ]], ⇒
SCROLL.target_scrollname
```

The target scroll level in this construction is the scroll level that you want to specify. If the target level is one, you need to supply only the **SCROLL.***target_scrollname* parameter. If the target scroll level is greater than one, you need to provide scroll name and row-level parameters for hierarchically superior scroll levels, beginning at level one. The following table indicates the scroll path syntax for the three possible target scroll levels:

Target Level	Scroll Path Syntax
1	`SCROLL.target_scrollname`
2	`SCROLL.level1_scrollname, level1_row, SCROLL.target_⇒ scrollname`
3	`SCROLL.level1_scrollname, level1_row, SCROLL.level2_⇒ scrollname, level2_row, SCROLL.target_scrollname`

If the component you are referring to is a row or a buffer field, additional parameters are required after the scroll path.

The following are the alternative scroll path syntax parameters:

Parameter	Description
SCROLL.*level1_scrollname*	Specifies the name of the page's level-one scroll area. This is always the same as the name of the scroll level's primary scroll record. This parameter is required if the target scroll level is two or three.
level1_row	An integer that selects a row on scroll level one. This parameter is required if the target scroll level is two or three.
SCROLL.*level2_scrollname*	Specifies the name of the page's level two scroll area. This is always the same as the name of the scroll level's primary scroll record. This parameter is required if the target row is on scroll level three.
level2_row	An integer that selects a row on scroll level two. This parameter is required if the target row is on scroll level three.
SCROLL.*target_scrollname*	The scroll name of the target scroll level, which can be level one, two, or three of the active page.

See Also

Chapter 4, "Referencing Data in the Component Buffer," Referencing Scroll Levels, Rows, and Buffer Fields, page 52

Referencing Scroll Levels, Rows, and Buffer Fields

You can reference a scroll level using the *scrollpath* construct only. Functions that reference rows for buffer fields require additional parameters. The following table summarizes the three types of component buffer references:

Target Component	Reference Syntax	Example Function
Scroll level	`scrollpath`	`HideScroll(scrollpath);`
Row	`scrollpath, row_number`	`HideRow(scrollpath, row_⇒ number);`
Field	`scrollpath, row_number, [recordname.]fieldname`	`FetchValue(scrollpath, row_⇒ number, fieldname);`

PeopleTools 8 provides an alternative to the scroll level, row, and field components in the form of the data buffer classes Rowset, Row, Record, and Field, which you reference using dot notation with object methods and properties. The following table demonstrates the syntax for instantiating and manipulating objects in the current context from these classes:

Target Object	Example Instantiation	Example Operation
Rowset	`&MYROWSET = GetRowset();`	`&MYROWSET.Refresh();`
Row	`&MYROW = GetRow();`	`&MYROW.CopyTo(&SOMEROW);`
Record	`&MYRECORD = GetRecord();`	`&MYREC.CompareFields(&REC);`
Field	`&MYFIELD = GetRecord(). fieldname;`	`&MYFIELD.Label = "Last Name";`

The following sections provide examples of functions using scroll path syntax, which refer to an example page from a fictitious veterinary clinic database. The page has three scroll levels, shown in the following table:

Level	Scroll Name (Primary Scroll Record Name)
0	VET
1	OWNER
2	PET
3	VISIT

The examples given for PeopleTools 8 object-oriented syntax assumes that the following initializing code has been executed:

```
Local Rowset  &VET_SCROLL, &OWNER_SCROLL, &PET_SCROLL, &VISIT_SCROLL;

&VET_SCROLL = GetLevel0();
&OWNER_SCROLL = &VET_SCROLL.GetRow(1).GetRowSet(SCROLL.OWNER);
&PET_SCROLL = &OWNER_SCROLL.GetRow(2).GetRowSet(SCROLL.PET);
&VISIT_SCROLL = &PET_SCROLL.GetRow(2).GetRowSet(SCROLL.VISIT);
```

Referring to Scroll Levels

The HideScroll function provides an example of a reference to a scroll level. The syntax of the function is:

HideScroll(*scrollpath*)

where *scrollpath* is

[**RECORD.***level1_recname*, *level1_row*, [**RECORD.***level2_recname*, *level2_row*,]] **RECORD.**⇒ *target_recname*

To reference the level 1 scroll in the example, use this syntax:

```
HideScroll(RECORD.OWNER);
```

This hides the OWNER, PET, and VISIT scroll areas on the example page.

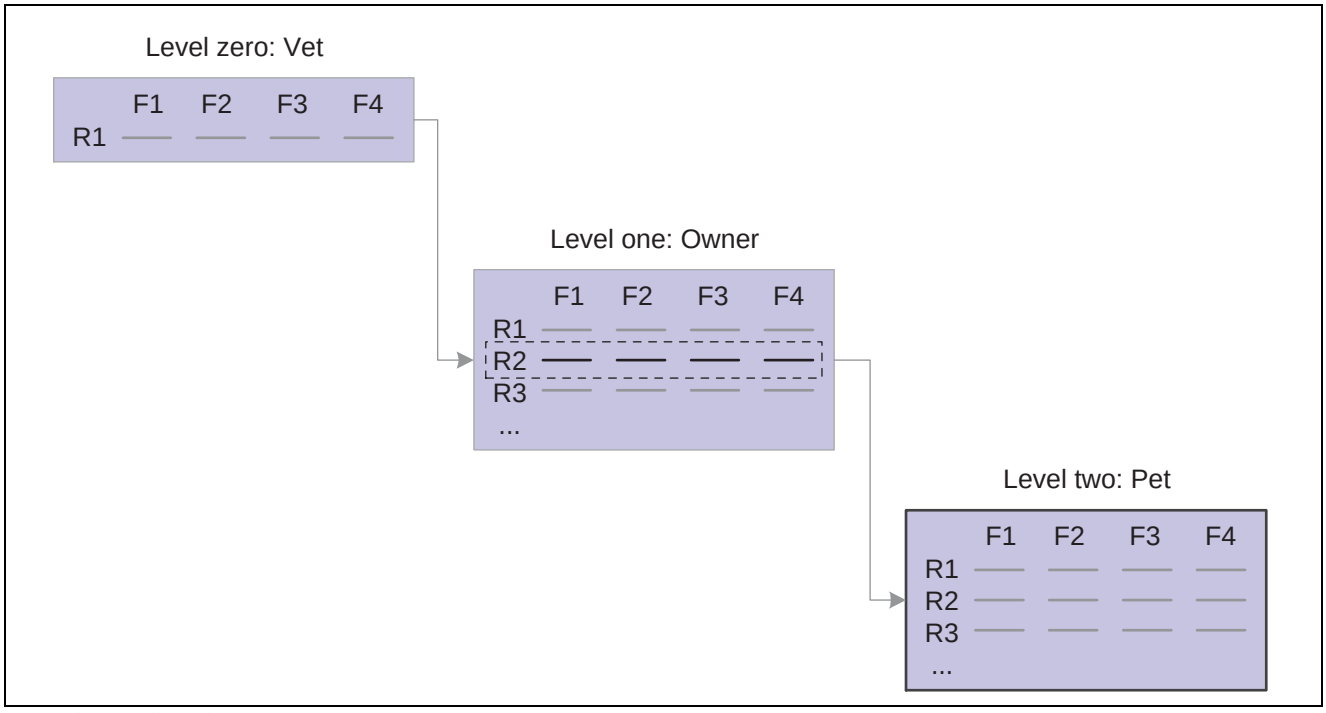
In PeopleTools 8, the object-oriented version of this is:

```
&OWNER_SCROLL.HideAllRows();
```

To hide scroll levels two and below, supply the primary record and row in scroll level one, then the record identifying the target scroll area:

```
HideScroll(RECORD.OWNER, &L1ROW, RECORD.PET);
```

The following diagram indicates the scroll path of this statement, assuming that the value of &L1ROW is 2:



Sample scroll path

Similarly, to hide the VISIT scroll area on level three, you specify rows on scroll levels one and two.

```
HideScroll(RECORD.OWNER, &L1ROW, RECORD.PET, &L2ROW, RECORD.VISIT);
```

To use the **SCROLL**.*scrollname* syntax, the previous example could be written as the following:

```
HideScroll(SCROLL.OWNER, &L1ROW, SCROLL.PET, &L2ROW, SCROLL.VISIT);
```

In PeopleTools 8, the object-oriented version of this is:

```
&VISIT_SCROLL.HideAllRows();
```

Referring to Rows

Referring to rows is the same as referring to scroll areas, except that you need to specify the row you want to select on the target scroll area. As an example, let's examine the HideRow function, which hides a specific row in the level three scroll area of the page. Here is the function syntax:

HideRow(*scrollpath, target_row*)

To hide row number &ROW_NUM on level one:

```
HideRow(RECORD.OWNER, &ROW_NUM);
```

To do the same using the **SCROLL**.*scrollname* syntax:

```
HideRow(SCROLL.OWNER, &ROW_NUM);
```

In PeopleTools 8, the object-oriented version of this for the OWNER rowset is:

```
&OWNER_SCROLL(&ROW_NUM).Visible = False;
```

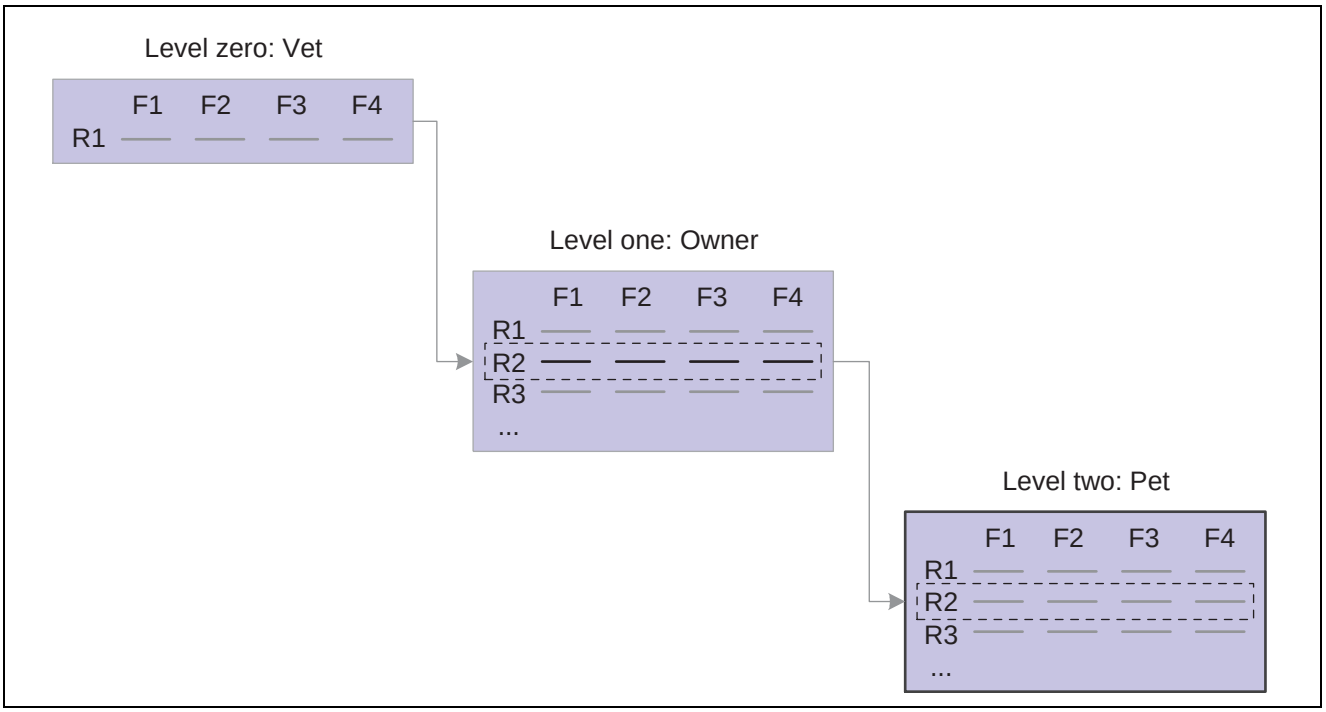
On level two:

```
HideRow(RECORD.OWNER, &L1_ROW), RECORD.PET, &ROW_NUM);
```

In PeopleTools 8, the object-oriented version of this for the PET rowset is:

```
&PET_SCROLL(&ROW_NUM).Visible = False;
```

The following diagram indicates the scroll path of this statement, assuming that the value of &L1_ROW is 2 and that &ROW_NUM is equal to 2:



Scroll path statement

On level three:

```
HideRow(RECORD.OWNER, CurrentRowNumber(1), RECORD.PET,
CurrentRowNumber(2), RECORD.VISIT, &ROW_NUM);
```

In PeopleTools 8, the object-oriented version of this for the VISIT rowset is:

```
&VISIT_SCROLL(&ROW_NUM).Visible = False;
```

Referring to Buffer Fields

Buffer field references require a [*recordname.*]*fieldname* parameter to specify a record field. The combination of scroll level, row number, and record field name uniquely identifies the buffer field. Here is the syntax:

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

Assume, for example, that record definitions in the veterinary database have the following fields that you want to reference:

Record	Sample Field
OWNER	OWNER_NAME
PET	PET_BREED
VISIT	VISIT_REASON

You could use the following examples to retrieve values on levels one, two, or three from a PeopleCode program executing on level zero.

To fetch a value of the OWNER_NAME field on the current row of scroll area one:

```
&SOMENAME = FetchValue(RECORD.OWNER, &L1_ROW, OWNER.OWNER_NAME);
```

In PeopleTools 8, the object-oriented version of this for the OWNER rowset is:

```
&SOMENAME = &OWNER_SCROLL(&L1_ROW).OWNER.OWNER_NAME;
```

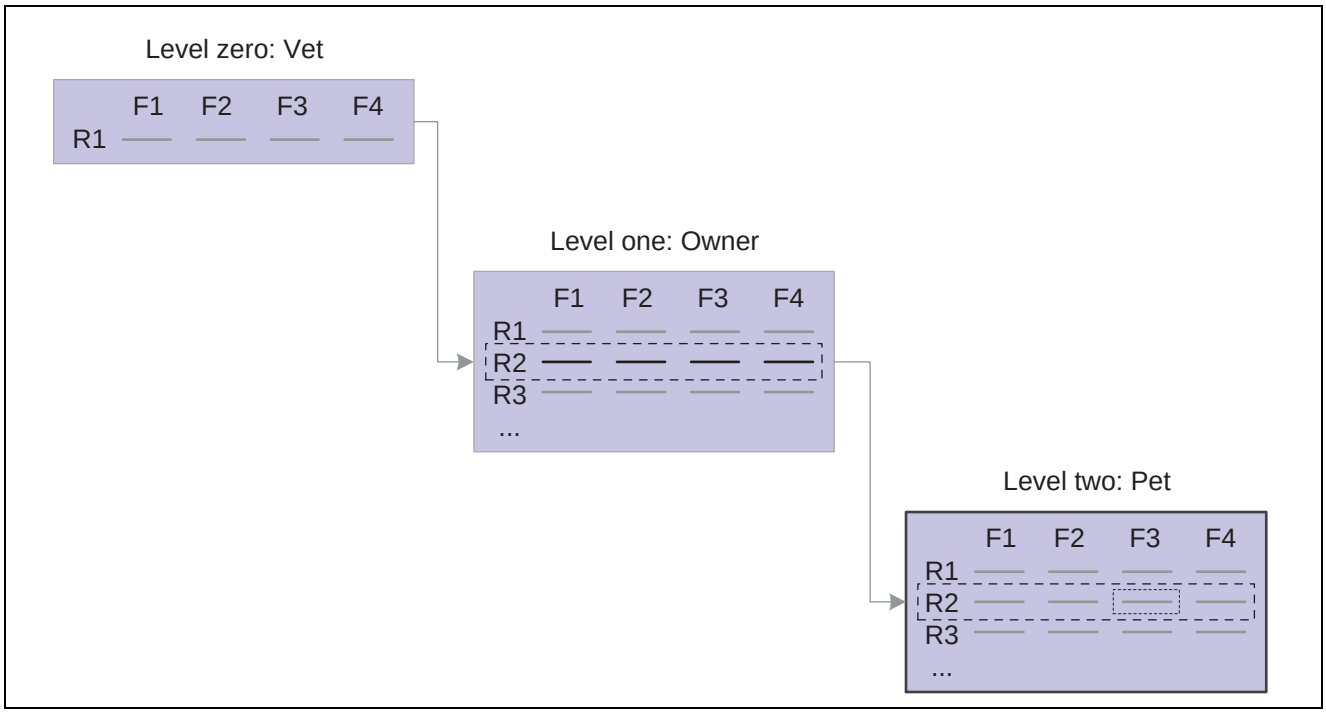
To fetch PET_BREED on level two:

```
&SOMEBREED = FetchValue(RECORD.OWNER, &L1_ROW, RECORD.PET, &L2_ROW, PET.PET_BREED);
```

In PeopleTools 8, the object-oriented version of this for the PET rowset is:

```
&SOMEBREED = &PET_SCROLL(&L2_ROW).PET.PET_BREED;
```

The following diagram indicates the scroll path to the target field, assuming that &L1_ROW equals 2, &L2_ROW equals 2, and field F3 is PET.PET_BREED:



Scroll path to target field

To fetch VISIT_REASON on level three:

```
&SOMEREASON = FetchValue(RECORD.OWNER, &L1_ROW, RECORD.PET,
&L2_ROW, RECORD.VISIT, &L3_ROW, VISIT.VISIT_REASON);
```

To do the same using the **SCROLL**.*scrollname* syntax:

```
&SOMEREASON = FetchValue(SCROLL.OWNER, &L1_ROW, SCROLL.PET,
&L2_ROW, SCROLL.VISIT, &L3_ROW, SCROLL.VISIT_REASON);
```

In PeopleTools 8, the object-oriented version of this is:

```
&SOMEREASON = &VISIT_SCROLL(&L3_ROW).VISIT.VISIT_REASON;
```

Using CurrentRowNumber

The CurrentRowNumber function returns the current row, as determined by the current context, for a specific scroll level in the active page. CurrentRowNumber is often used to determine a value for the *level1_row* and *level2_row* parameters in scroll path constructions. Because current row numbers are determined by the current context, CurrentRowNumber cannot determine a current row on a scroll level outside the current context (a scroll level below the level where the PeopleCode program is currently executing).

For example, you could use a statement like this to retrieve the value of a buffer field on level three of the PET_VISITS page, in a PeopleCode program executing on level two:

```
&VAL = FetchValue(RECORD.OWNER, CurrentRowNumber(1),
RECORD.PET, CurrentRowNumber(2), RECORD.VISIT, &TARGETROW,
VISIT_REASON);
```

Because the PeopleCode program is executing on level two, CurrentRowNumber can return values for levels one and two, but not three, because level three is outside of the current context and has no current row number.

Looping Through Scroll Levels

Component buffer functions are often used in For loops to loop through the rows on scroll levels below the level where the PeopleCode program is executing. The following loop, for example could be used in PeopleCode executing on a level two record field to loop through rows of data on level three:

```
For &I = 1 To ActiveRowCount(RECORD.OWNER,
CurrentRowNumber(1), RECORD.PET, CurrentRowNumber(2), RECORD.VISIT)
    &VAL = FetchValue(RECORD.OWNER, CurrentRowNumber(1),
RECORD.PET, CurrentRowNumber(2), RECORD.VISIT, &I, VISIT_REASON);
   If &VAL = "Fleas" Then
      /* do something about fleas */
   End-If;
End-For;
```

A similar construct may be used in accessing other level two or level one scroll areas, such as work scroll areas.

In these constructions, the ActiveRowCount function is often used to determine the upper bounds of the loop. When ActiveRowCount is used for this purpose, the loop goes through all of the active rows in the scroll (rows that have not been specified as deleted). If you use TotalRowCount to determine the upper bounds of the loop, the loop goes through all of the rows in the scroll area: first those that have not been specified as deleted, then those that have been specified as deleted.

See Also

Chapter 4, “Referencing Data in the Component Buffer,” Structuring Scroll Path Syntax in PeopleTools 7.5, page 49

Chapter 4, “Referencing Data in the Component Buffer,” Understanding Current Context, page 45

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” CurrentRowNumber

CHAPTER 5

Accessing the Data Buffer

This chapter provides overviews of data buffer classes, data buffer class examples, and current context, and discusses how to:

- Access secondary component buffer data.
- Instantiate rowsets using non-component buffer data.

Understanding Data Buffer Access

This section discusses:

- Data buffer access.
- Access classes.
- Data buffer model and data access objects.

Data Buffer Access

In addition to the built-in functions you use to access the component buffer, there are classes of objects that provide access to structured data buffers using the PeopleCode object syntax.

The data buffers accessed by these classes are typically the component buffers that are loaded when you open a component. However, these classes may also be used to access data from general data buffers, loaded by an application message, an Application Engine program, a component interface, and so on.

The methods and properties of these classes provide functionality that is similar to what has been available using built-in functions. However, they also provide improved consistency, flexibility, and new functionality.

Access Classes

There are four data buffer classes: Rowset, Row, Record, and Field. These four classes are the foundation for accessing component buffer data through the new object syntax.

A field object, which is instantiated from the Field class, is a single instance of data within a record and is based on a field definition.

A record object, which is instantiated from the Record class, is a single instance of a data within a row and is based on a record definition. A record object consists of one to *n* fields.

A row object, which is instantiated from the Row class, is a single row of data that consists of one to *n* records of data. A single row in a component scroll area is a row. A row may have one to *n* child rowsets. For example, a row in a level two scroll area may have *n* level three child rowsets.

A rowset object is a data structure used to describe hierarchical data. It is made up of a collection of rows. A component scroll area is a rowset. You can also have a level zero rowset.

Data Buffer Model and Data Access Classes

The data model assumed by the data buffer classes is that of a PeopleTools component, where scroll bars or grids are used to describe a hierarchical, multiple-occurrence data structure. You can access these classes using dot notation.

The four data buffer classes relate to each other in a hierarchical manner. The main points to understand these relationships are:

- A record contains one or more fields.
- A row contains one or more records and zero or more child rowsets.
- A rowset contains one or more rows.

For component buffers, think of a rowset as a scroll area on a page that contains all of that scroll area's data. A level zero rowset contains all the data for the entire component. You can use rowsets with application messages, file layouts, business interlinks, and other definitions besides components. A level zero rowset from a component buffer only contains one row, the keys that the user specifies to initiate that component. A level zero rowset from data that is not a component, such as a message or a file layout, might contain more than one level zero row.

The following is some basic PeopleCode that traverses through a two-level component buffer using dot notation syntax. Level zero is based on record QA_INVEST_HDR, and level one is based on record QA_INVEST_LN.

```
Local Rowset &HDR_ROWSET, &LINE_ROWSET;
Local Record &HDR_REC, &LINE_REC;
&HDR_ROWSET = GetLevel0();

For &I = 1 to &HDR_ROWSET.RowCount
   &HDR_REC = &HDR_ROWSET(&I).QA_INVEST_HDR;
   &EMPLID = &HDR_REC.EMPLID.Value;
   &LINE_ROWSET = &HDR_ROWSET(&I).GetRowset(1);
   For &J = 1 to &LINE_ROWSET.RowCount
      &LINE_REC = &LINE_ROWSET(&J).QA_INVEST_LN;
      &LINE_SUM = &LINE_SUM + &LINE_REC.AMOUNT.Value;
   End-For;
End-For;
```

Each rowset is declared and instantiated. In general, your code is easier to read and maintain if you follow this practice.

Understanding Data Buffer Classes Examples

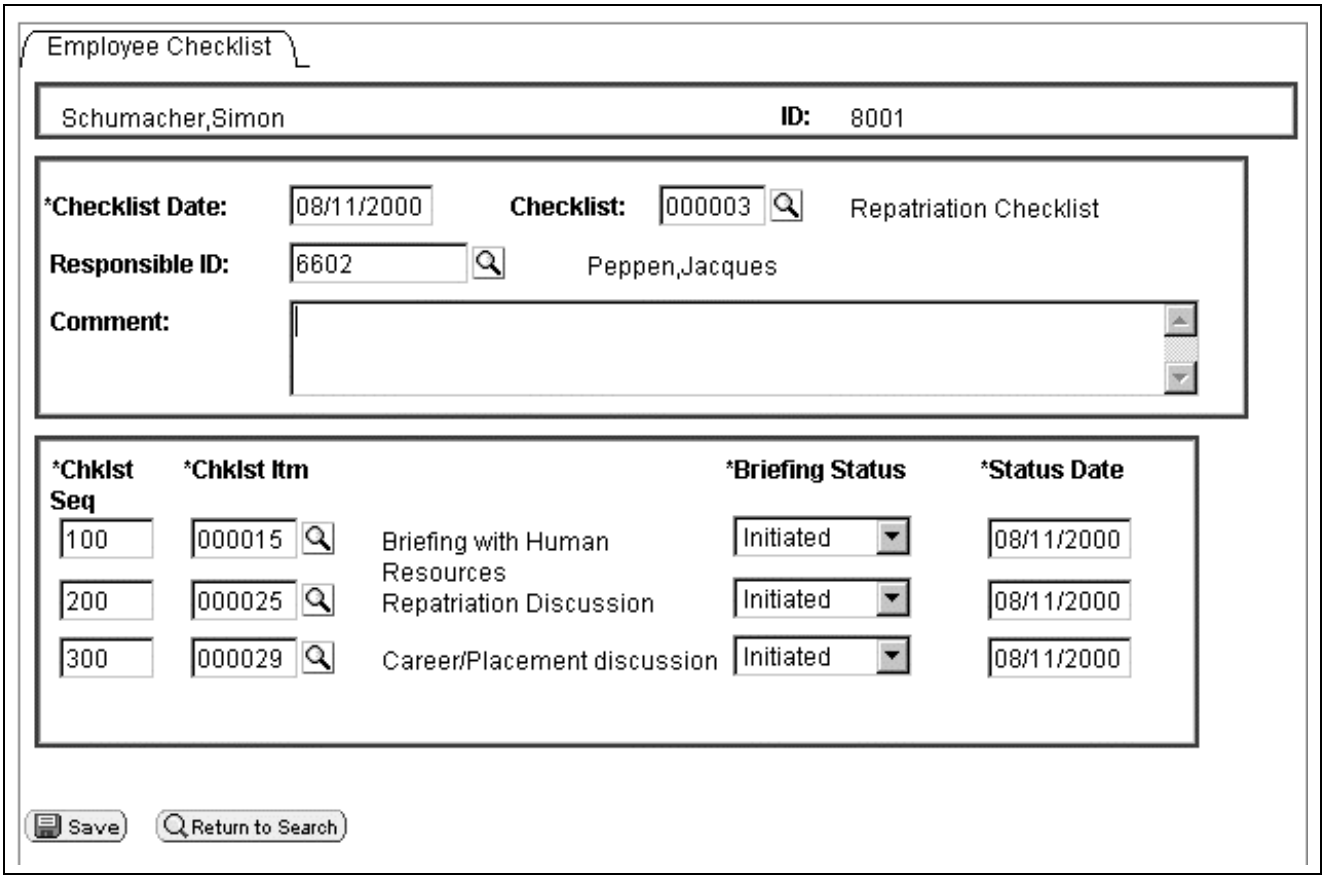
This section discusses:

- Employee Checklist page structure.
- Object creation examples.
- Data buffer hierarchy examples.

- Rowset examples.
- Hidden work scroll area examples.

Employee Checklist Page Structure

Most of the examples in this section use the Employee Checklist page.

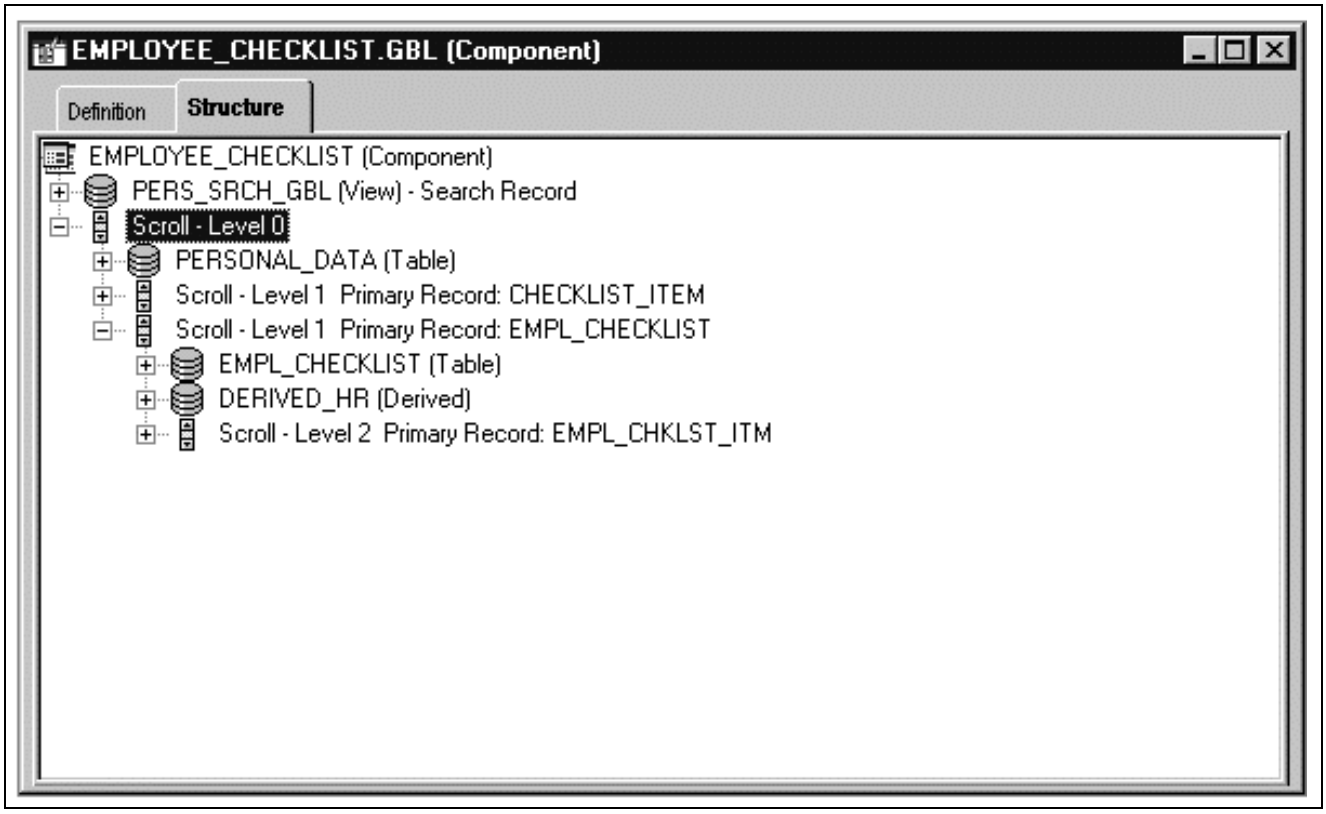


Employee Checklist page

This page has the following record structure:

Scroll Level	Associated Primary Record	Rowset and Variable Name
Level zero	PERSONAL_DATA	Level zero rowset: &RS0
Level one scroll area	EMPL_CHECKLIST	Level one rowset: &RS1
Level one hidden work scroll area	CHECKLIST_ITEM	Level one rowset: &RS1H
Level two scroll area	EMPL_CHKLST_ITM	Level two rowset: &RS2

Another way of looking at the structure of a component is to use the Structure view. All the scroll areas are labeled, and the primary record is associated with each:

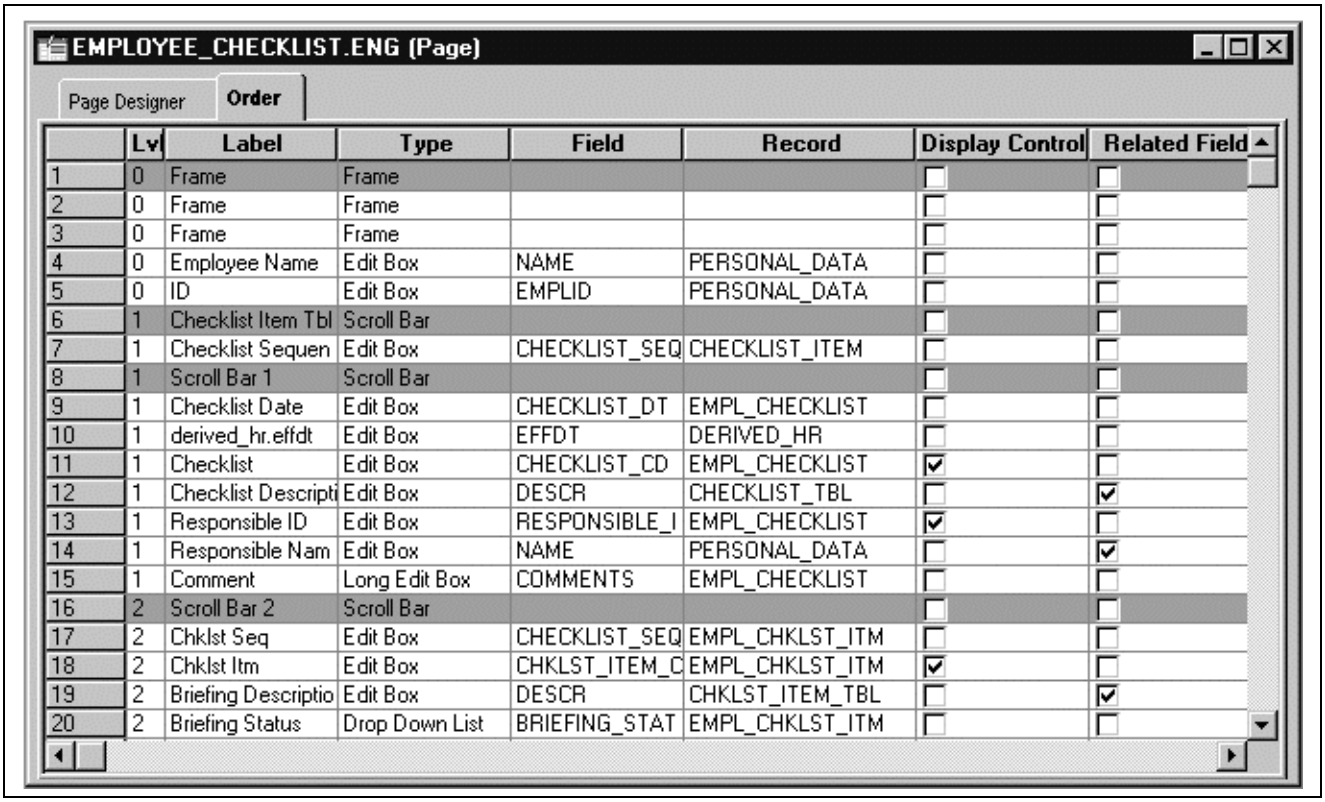


EMPLOYEE_CHECKLIST structure

In the example, the visible level one scroll area also has only one row. That row is made up of the following records:

- EMPL_CHECKLIST
- DERIVED_HR
- CHECKLIST_TBL
- PERSONAL_DATA

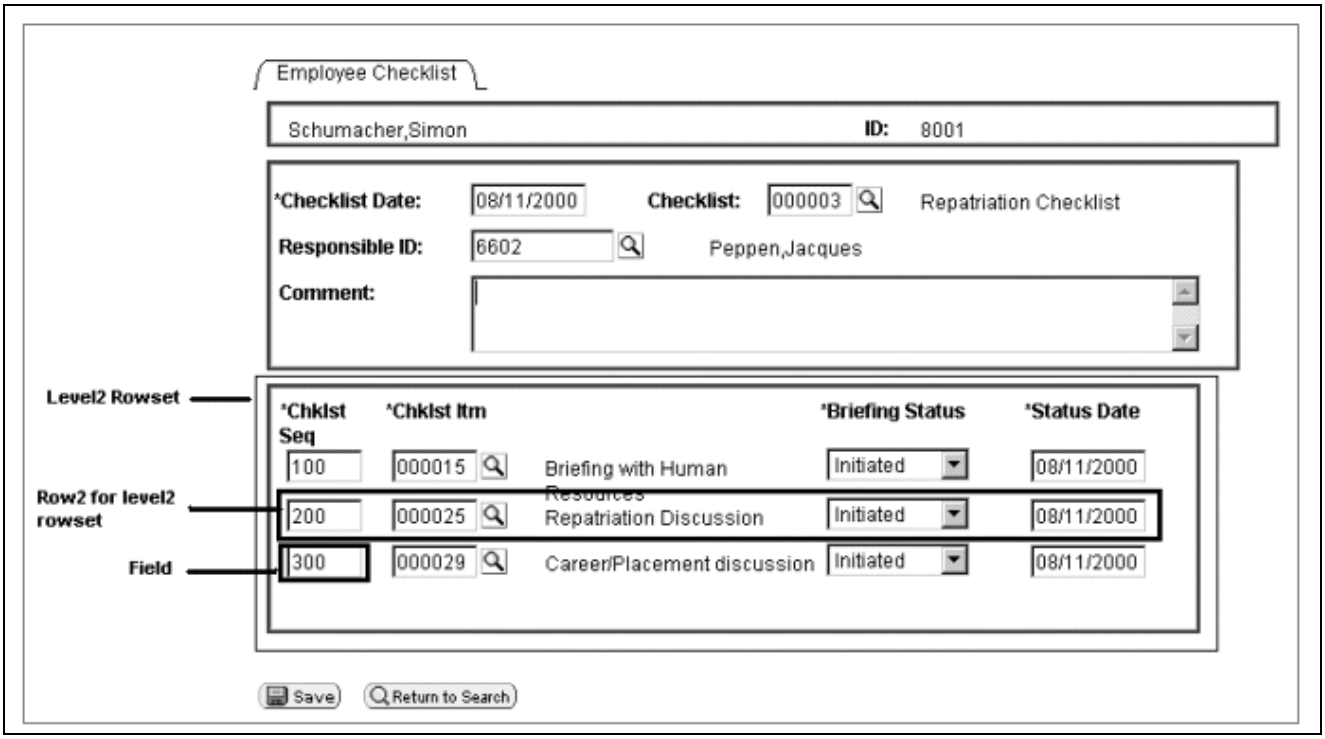
You can see which records are associated with a scroll are by looking at the Order view for a page:



	Lvl	Label	Type	Field	Record	Display Control	Related Field
1	0	Frame	Frame			☐	☐
2	0	Frame	Frame			☐	☐
3	0	Frame	Frame			☐	☐
4	0	Employee Name	Edit Box	NAME	PERSONAL_DATA	☐	☐
5	0	ID	Edit Box	EMPLID	PERSONAL_DATA	☐	☐
6	1	Checklist Item Tbl	Scroll Bar			☐	☐
7	1	Checklist Sequen	Edit Box	CHECKLIST_SEQ	CHECKLIST_ITEM	☐	☐
8	1	Scroll Bar 1	Scroll Bar			☐	☐
9	1	Checklist Date	Edit Box	CHECKLIST_DT	EMPL_CHECKLIST	☐	☐
10	1	derived_hr.effdt	Edit Box	EFFDT	DERIVED_HR	☐	☐
11	1	Checklist	Edit Box	CHECKLIST_CD	EMPL_CHECKLIST	☑	☐
12	1	Checklist Descripti	Edit Box	DESCR	CHECKLIST_TBL	☐	☑
13	1	Responsible ID	Edit Box	RESPONSIBLE_I	EMPL_CHECKLIST	☑	☐
14	1	Responsible Nam	Edit Box	NAME	PERSONAL_DATA	☐	☑
15	1	Comment	Long Edit Box	COMMENTS	EMPL_CHECKLIST	☐	☐
16	2	Scroll Bar 2	Scroll Bar			☐	☐
17	2	Chklst Seq	Edit Box	CHECKLIST_SEQ	EMPL_CHKLST_ITM	☐	☐
18	2	Chklst Itm	Edit Box	CHKLST_ITEM_C	EMPL_CHKLST_ITM	☑	☐
19	2	Briefing Descriptio	Edit Box	DESCR	CHKLST_ITEM_TBL	☐	☑
20	2	Briefing Status	Drop Down List	BRIEFING_STAT	EMPL_CHKLST_ITM	☐	☐

EMPLOYEE_CHECKLIST page Order view showing records

The level two rowset has three rows. Each row is made up of two records: the primary record, EMPL_CHKLST_ITM, and CHKLST_ITM_TBL, the record associated with the related display field DESCR. The following illustration shows the rowset:



EMPLOYEE_CHECKLIST rowsets and rows

Every record has fields associated with it, such as NAME, EMPLID and CHECKLIST_SEQ. These are the fields associated with the record definitions, not the fields displayed on the page.

Object Creation Examples

When declaring variables, using the class with the same name as the data buffer access data type (rowset objects should be declared as type Rowset, field objects as type Field, and so on). Data buffer access class objects can be of type Local, Global, or Component.

The following declarations are assumed throughout the examples that follow:

```
Local Rowset &LEVEL0, &ROWSET;
Local Row &ROW;
Local Record &REC;
Local Field &FIELD;
```

Level Zero Access

The following code instantiates a rowset object, from the Rowset class, which references the level zero rowset, containing all the page data. It stores the object in the &LEVEL0 variable.

```
&LEVEL0 = GetLevel0();
```

The level zero rowset contains all the rows, rowsets, records, and fields underneath it.

If the level zero rowset is formed from component buffer data, the level zero rowset has one row of data, and that row contains all the child rowsets, which in turn contain rows of data that contain other child rowsets.

If the level zero rowset is formed from buffer data, such as from an application message, the level zero rowset may contain more than one row of data. Each row of the level zero rowset contains all the child rowsets associated with that row, which in turn contain rows of data that contain other child rowsets.

Use a level zero rowset when you want an absolute path to a lower-level object or to do some processing on the entire data buffer. For example, suppose you load all new data into the component buffers and want to redraw the page. You could use the following code:

```
/* Do processing to reload Component Buffers */
&LEVEL0 = GetLevel0();
&LEVEL0.Refresh();
```

Rowset Object

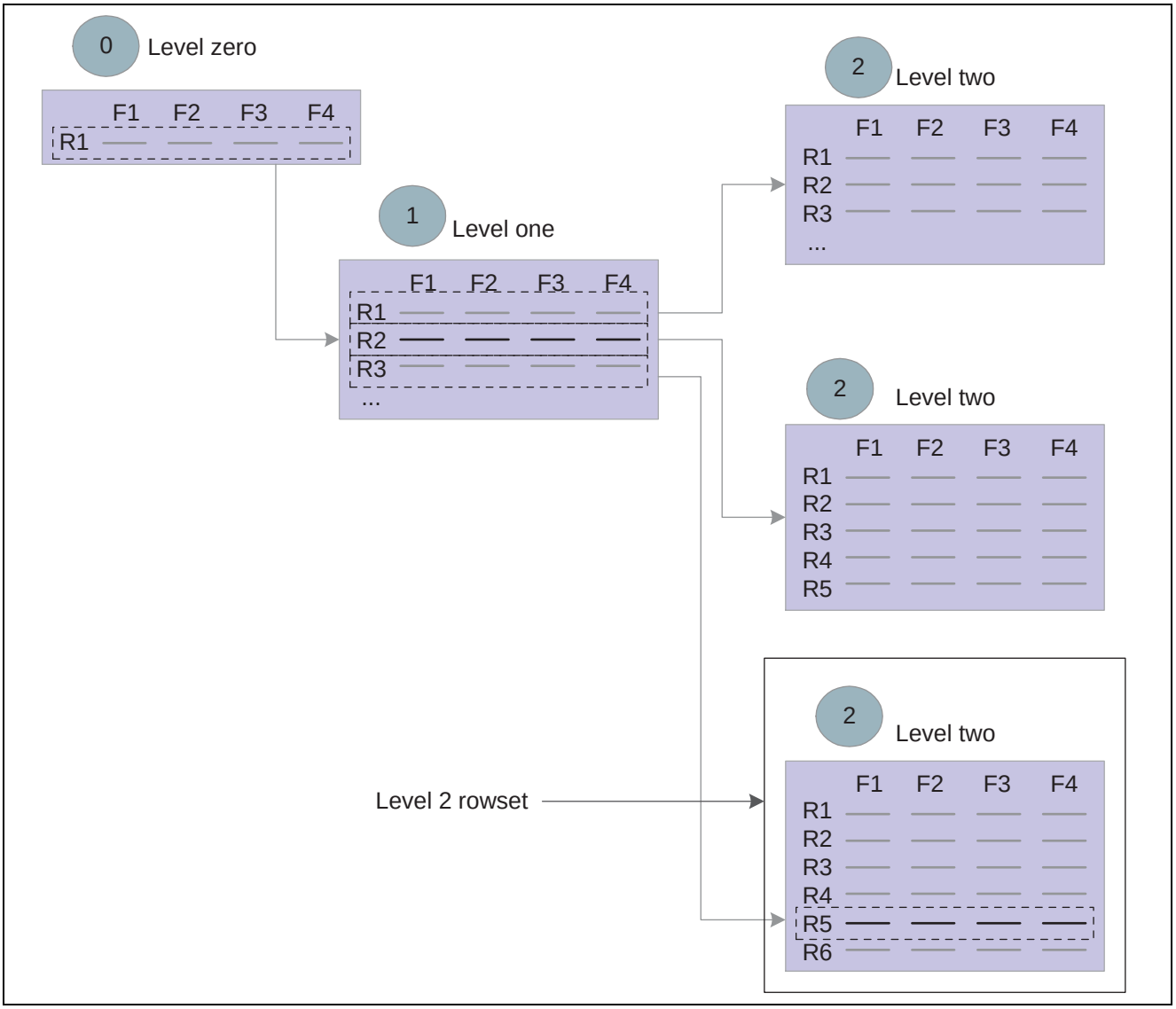
The following code instantiates a rowset object that references the rowset that contains the currently running PeopleCode program.

```
&ROWSET = GetRowset();
```

You might later use the &ROWSET variable and the ActiveRowCount property to iterate over all the rows of the rowset, or to access a specific row (using the GetRow method) or to hide a child rowset (by setting the Visible property).

The level one rowset contains all the level two rowsets. However, the level two rowsets can only be accessed using the different rows of the level one rowset. From the level zero or level one rowset, you can only access a level two rowset by using the level one rowset and the appropriate row.

For example, suppose your program is running on some field of row five of a level two scroll area, which is on row three of its level one scroll area. The resulting rowset contains all the rows of the level two scroll area that are under the row three of the level one scroll area. The rowset does not contain any data that is under any other level two scroll areas. The following diagram illustrates these results:

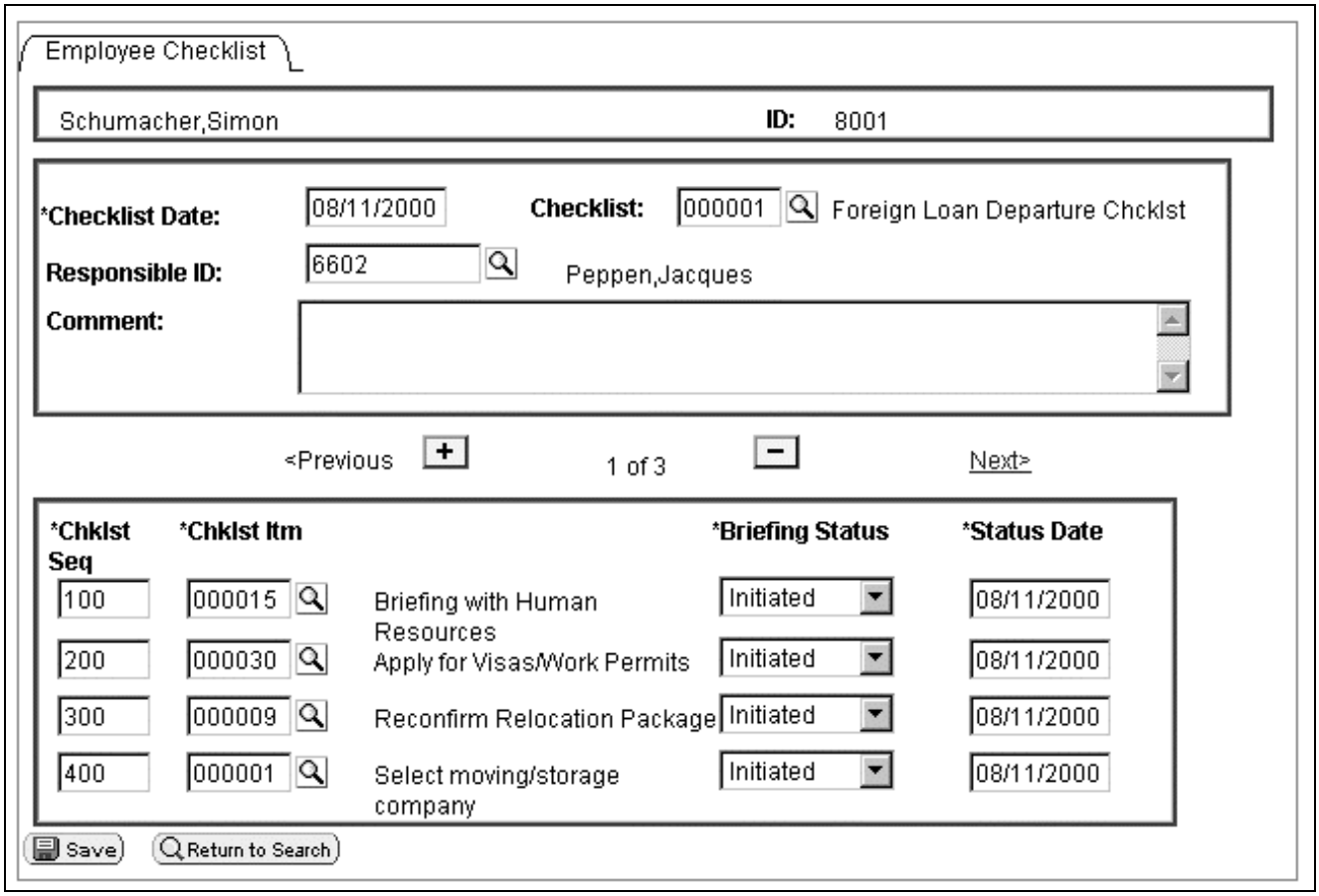


Level two rowset from level one row

Let's illustrate this further with an example from the Employee Checklist page.

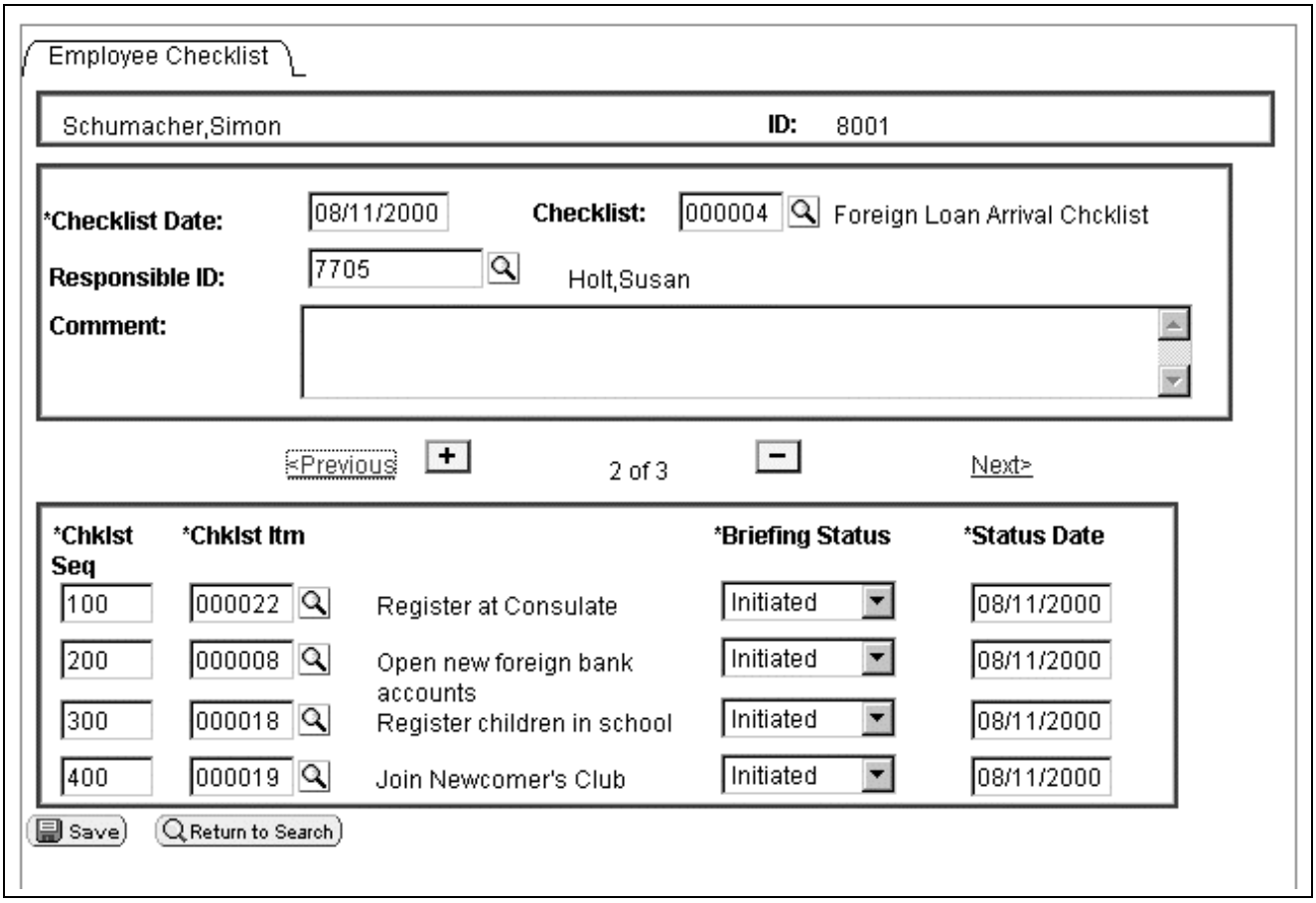
Suppose that one employee was associated with three different checklists: Foreign Loan Departure, Foreign Loan Arrival, and Foreign Loan Host. The checklist code field (CHECKLIST_CD) on the first level of the page drives the entries on the second level. Each row in the level one rowset produces a different level two rowset.

The Foreign Loan Departure checklist (000001) produces a checklist that contains such items as Briefing with Human Resources and Apply for Visas/Work permits, as shown in the following illustration:



EMPLOYEE_CHECKLIST Foreign Loan Departure checklist

The Foreign Loan Arrival checklist (0000004) produces a checklist that contains items such as Register at Consulate and Open New Foreign Bank Accounts, as shown in the following illustration:



EMPLOYEE_CHECKLIST Foreign Load Arrival Checklist

Row Object

When you create a page, you put fields from different records onto the page. You can think of this as creating a type of pseudo-Structured Query Language (SQL) join. The row returned from this pseudo-join is a row object.

For example, the first level scroll area of the EMPLOYEE_CHECKLIST page contains the following fields, associated with these records:

Field	Record
CHECKLIST_DT	EMPL_CHECKLIST
CHECKLIST_CD	EMPL_CHECKLIST
COMMENTS	EMPL_CHECKLIST
DESCR	CHECKLIST_TBL

Field	Record
NAME	PERSONAL_DATA
RESPONSIBLE_ID	EMPL_CHECKLIST

The pseudo-SQL join might look like the following:

```
JOIN A.CHECKLIST_DT, A.CHECKLIST_CD, A.COMMENTS, B.DESCR, C.NAME, A.RESPONSIBLE_ID
FROM PS_EMPL_CHECKLIST A, PS_CHECKLIST_TBL B, PS_PERSONAL_DATA C, WHERE. . .
```

What goes into the Where clause is determined by the level zero of the page. For our example, it's `WHERE EMPLID=8001.`

When the component is opened, data gets loaded into the component buffers. Any row returned by the pseudo-SQL statement is a level one row object. The following table illustrates a returned row:

CHECKLIST_DT	CHECKLIST_CD	COMMENTS	DESCR	NAME	RESPONSIBLE_ID
12/03/98	000001		Foreign Loan Department Checklist	Peppen, Jacques	6602

Record Object

A record definition is a definition of what your underlying SQL database tables look like and how they process data. After you create record definitions, you build the underlying SQL tables that contain the application data your users enter online in your production environment.

When you create a record object using the CreateRecord function, you're creating an area in the data buffers that has the same structure as the record definition, but no data.

When you instantiate a record object from the Record class using some variation of GetRecord, that record object references a single row of data in the SQL table.

Note. The data in the record that you retrieve is based on the row. This is analogous to setting keys to return a unique record.

The following code instantiates a record object for referencing the EMPL_CHECKLIST record of the specified row.

```
&REC = &ROW.GetRecord(RECORD.EMPL_CHECKLIST);
```

Using the short method, the following line of code is identical to the previous line:

```
&REC = &ROW.EMPL_CHECKLIST;
```

You might later use the &REC variable and the CopyFieldsTo property to copy all like-named fields from one record to another. In the following example, two row objects are created, the copy from row (COPYFRMROW) and the copy to row (COPYTROW). Using these rows, like-named fields are copied from CHECKLIST_ITEM to EMPL_CHKLST_ITM.

```
For &I = 1 To &ROWSET1.ActiveRowCount
   &COPYFRMROW = &ROWSET1.GetRow(&I);
   &COPYTROW = &RS2.GetRow(&I);
```

```
   &COPYFRMROW.CHECKLIST_ITEM.CopyFieldsTo(&COPYTROW.EMPL_CHKLST_ITM);
End-For;
```

A row may contain more than one record: in addition to the primary database record, you may have a related display record or a derived record. You can access these records as well. The level one rowset, &ROWSET1, is made up of many records. The following accesses two of them: EMPL_CHECKLIST and DERIVED_HR.

```
&REC1 = &ROW.EMPL_CHECKLIST;
&REC2 = &ROW.DERIVED_HR;
```

Field Object

The following instantiates a field object, from the Field class, that is used to access a specific field in the record.

```
&FIELD = &REC.GetField(FIELD.CHECKLIST_CD);
```

You might later use the &FIELD variable as a condition:

```
If ALL(&FIELD) Then
```

Here is another example:

```
If &FIELD.Value = "N" Then
```

Note. The data in the field that you retrieve is based on the record, which is in turn based on the row.

You can also set the value of a field. Using the GetField function does not create a copy of the data from the component buffer. Setting the value or a property of the field object sets the actual component buffer field or property.

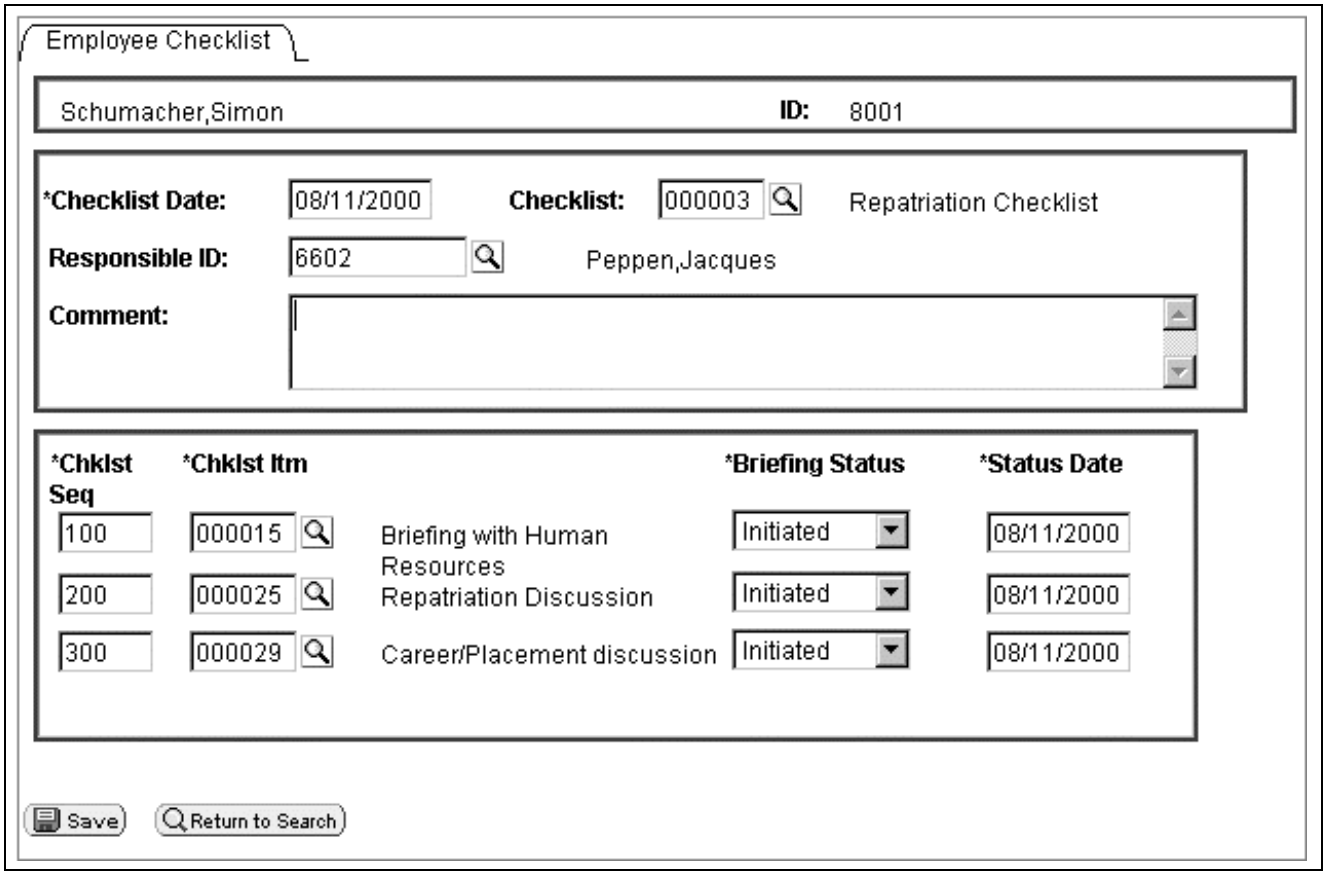
See Chapter 3, “Understanding Objects and Classes in PeopleCode,” Assigning Objects, page 38.

In the following example, the type of a field is checked, and the value is replaced with the tangent of that value if it is a number.

```
If &FIELD.Type <> "NUMBER" Then
   /* do error recording */
Else
   &FIELD.Value = Tan(&FIELD.Value);
End-If;
```

Data Buffer Hierarchy Examples

Suppose you want to access the BRIEFING_STATUS field at level two of the following page:



EMPLOYEE_CHECKLIST repatriation checklist

First, determine where your code is running. For this example, the code is starting at a field on a record at level zero. However, you do not always have to start at level zero.

After you start with level zero, you must traverse the data hierarchy, through the level one rowset, to the level two rowset, before you can access the record that contains the field.

Obtaining the Rowset

You first obtain the level zero rowset, which is the PERSONAL_DATA rowset. However, you do not need to know the name of the level zero rowset to access it.

```
&LEVEL0 = GetLevel0();
```

Obtaining Rows

The next object to get is a row. As the following code is working with data that is loaded from a page, there is only one row at level zero. However, if you have rowsets that are populated with data that is not based on component buffers (for example, an application message) you may have more than one row at level zero.

```
&LEVEL0_ROW = &LEVEL0(1);
```

Obtaining Child Rowsets

To obtain the level two rowset, traverse through the level one rowset first. Therefore, the next object to get is the level one rowset, as shown in the following example:

```
&LEVEL1 = &LEVEL0_ROW.GetRowset(SCROLL.EMPL_CHECKLIST);
```

Obtaining Subsequent Rows

If you're traversing a page, obtain the appropriate row after you get a rowset. To process all the rows of the rowset, set this up in a loop, as shown in the following example:

```
For &I = 1 to &LEVEL1.ActiveRowCount
   &LEVEL1_ROW = &LEVEL1(&I);
   . . .
End-For;
```

Obtaining Subsequent Rowsets and Rows

Traverse another level in the page structure to access the second level rowset, and then use a loop to access the rows in the level two rowset.

Because we're processing all the rows at the level one, we're just adding code to the previous For loop. As we're processing through all the rows at level two, we're adding a second For loop. The new code is in bold in the following example:

```
For &I = 1 to &LEVEL1.ActiveRowCount
   &LEVEL1_ROW = &LEVEL1(&I);
   &LEVEL2 = &LEVEL1_ROW.GetRowset(SCROLL.
EMPL_CHKLST_ITM);
   For &J = 1 to &LEVEL2.ActiveRowCount
      &LEVEL2_ROW = &LEVEL2(&J);
   . . .
   End-For;
End-For;
```

Obtaining Records

A row always contains a record, and may contain only a child rowset, depending on how your page is set up. GetRecord is the default method for a row, so all you have to specify is the record name.

Because we're processing all the rows at level two, we're just adding code to the For loops of the previous example. The new code is in bold:

```
For &I = 1 to &LEVEL1.ActiveRowCount
   &LEVEL1_ROW = &LEVEL1(&I);
   &LEVEL2 = &LEVEL1_ROW.GetRowset(SCROLL.EMPL_CHKLST_ITM);
   For &J = 1 to &LEVEL2.ActiveRowCount
      &LEVEL2_ROW = &LEVEL2(&J);
      &RECORD = &LEVEL2_ROW.EMPL_CHKLST_ITM;
   . . .
   End-For;
End-For;
```

Obtaining Fields

Records are made up of fields. GetField is the default method for a record, so all you have to specify is the field name.

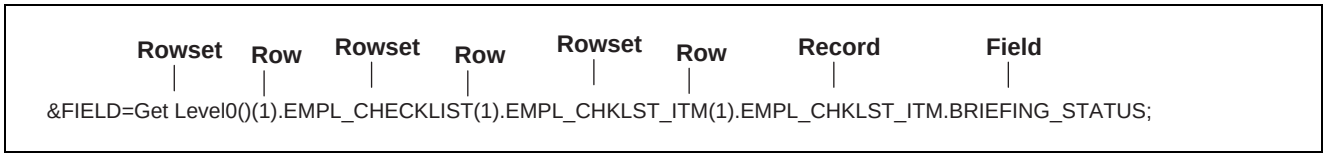
Because we're processing all the rows at the level one, we're just adding code to the For loops of the previous example. The new code is in bold:

```
For &I = 1 to &LEVEL1.ActiveRowCount
```

```
      &LEVEL1_ROW = &LEVEL1(&I);
      &LEVEL2 = &LEVEL1_ROW.GetRowset(SCROLL.EMPL_CHKLST_ITM);
      For &J = 1 to &LEVEL2.ActiveRowCount
         &LEVEL2_ROW = &LEVEL2(&J);
         &RECORD = &LEVEL2_ROW.EMPL_CHKLST_ITM;
         &FIELD = &RECORD.BRIEFING_STATUS;
         /* Do processing */
      End-For;
   End-For;
```

Using Shortcuts

The previous code is the long way of accessing this field. The following example uses shortcuts to access the field in one line of code. The following code assumes all rows are level one:



Rowset example

Here's another method of expressing the code:

Object Type	Code
Rowset	&LEVEL0 = GetLevel0();
Row	&LEVEL0_ROW = &LEVEL0(1);
Rowset	&LEVEL1 = &LEVEL0_ROW.GetRowset(SCROLL.EMPL_CHECKLIST);
	For &I = 1 to &LEVEL1.ActiveRowCount
Row	&LEVEL1_ROW = &LEVEL1(&I);
Rowset	&LEVEL2 = &LEVEL1_ROW.GetRowset(SCROLL.EMPL_CHKLST_ITM);
	For &J = 1 to &LEVEL2.ActiveRowCount
Row	&LEVEL2_ROW = &LEVEL2(&J);

Object Type	Code
Record	&RECORD = &LEVEL2_ROW.EMPL_CHKLST_ITM;
Field	&FIELD = &RECORD.BRIEFING_STATUS;
	/* Do processing */
	End-For;
	End-For;

Rowset Examples

The following code example traverses up to four levels of rowsets and could easily be modified to do more. This example only processes the first record in every rowset. To process every record, set up another For loop (For &R = 1 to &LEVELX.RECORDCOUNT, and so on.) Notice the use of the ChildCount function (to process all children rowsets within a rowset), ActiveRowCount, IsChanged, and dot notation.

In the following example, ellipses are used to indicate where application-specific code should go.

```
&Level0_ROWSET = GetLevel0();
For &A0 = 1 To &Level0_ROWSET.ActiveRowCount

    ...

/***************************/
/* Process Level 1 Records */
/*-------------------------*/
   If &Level0_ROWSET(&A0).ChildCount > 0 Then
   For &B1 = 1 To &Level0_ROWSET(&A0).ChildCount
      &LEVEL1_ROWSET = &Level0_ROWSET(&A0).GetRowset(&B1);
      For &A1 = 1 To &LEVEL1_ROWSET.ActiveRowCount
      If &LEVEL1_ROWSET(&A1).GetRecord(1).IsChanged Then

          ...

 /***************************/
 /* Process Level 2 Records */
 /*-------------------------*/
         If &LEVEL1_ROWSET(&A1).ChildCount > 0 Then
         For &B2 = 1 To &LEVEL1_ROWSET(&A1).ChildCount
            &LEVEL2_ROWSET = &LEVEL1_ROWSET(&A1).GetRowset(&B2);
            For &A2 = 1 To &LEVEL2_ROWSET.ActiveRowCount
            If &LEVEL2_ROWSET(&A2).GetRecord(1).IsChanged Then
```

```
                ...

/*************************/
/* Process Level 3 Records */
/*-------------------------*/
                If &LEVEL2_ROWSET(&A2).ChildCount > 0 Then
                For &B3 = 1 To &LEVEL1_ROWSET(&A2).ChildCount
                   &LEVEL3_ROWSET = &LEVEL2_ROWSET(&A2).GetRowset(&B3);
                   For &A3 = 1 To &LEVEL3_ROWSET.ActiveRowCount
                   If &LEVEL3_ROWSET(&A3).GetRecord(1).IsChanged Then

                       ...

                   End-If; /* A3 - IsChanged */
                   End-For; /* A3 - Loop */
                End-For; /* B3 - Loop */
                End-If; /* A2 - ChildCount > 0 */
/*--------------------------------*/
/* End of Process Level 3 Records */
/**********************************/

            End-If; /* A2 - IsChanged */
            End-For; /* A2 - Loop */
            End-For; /* B2 - Loop */
            End-If; /* A1 - ChildCount > 0 */
/*--------------------------------*/
/* End of Process Level 2 Records */
/**********************************/

      End-If; /* A1 - IsChanged */
      End-For; /* A1 - Loop */
   End-For; /* B1 - Loop */
   End-If; /* A0 - ChildCount > 0 */

/*--------------------------------*/
/* End of Process Level 1 Records */
/**********************************/

End-For; /* A0 - Loop */
```

Hidden Work Scroll Example

In the FieldChange event for the CHECKLIST_CD field on the EMPL_CHECKLIST record, a PeopleCode program does the following:

1. Flushes the rowset and hidden work scroll area.
2. Performs a Select statement on the hidden work scroll area based on the value of the CHECKLIST_CD field and the effective date.
3. Clears out the level two scroll area.

4. Copies like-named fields from the hidden work scroll area to the level two scroll area.

The following example shows how to do this using built-in functions.

```
&CURRENT_ROW_L1 = CurrentRowNumber(1);

&ACTIVE_ROW_L2 = ActiveRowCount(RECORD.EMPL_CHECKLIST,
&CURRENT_ROW_L1, RECORD.EMPL_CHKLST_ITM);

If All(CHECKLIST_CD) Then
   ScrollFlush(RECORD.CHECKLIST_ITEM);
   ScrollSelect(1, RECORD.CHECKLIST_ITEM, RECORD.CHECKLIST_ITEM,
"Where Checklist_Cd = :1 and EffDt = (Select Max(EffDt) From
PS_Checklist_Item Where Checklist_Cd = :2)",
CHECKLIST_CD, CHECKLIST_CD);

   &FOUNDDOC = FetchValue(CHECKLIST_ITEM.CHKLST_ITEM_CD, 1);
   &SELECT_ROW = ActiveRowCount(RECORD.CHECKLIST_ITEM);

   For &I = 1 To &ACTIVE_ROW_L2
      DeleteRow(RECORD.EMPL_CHECKLIST, &CURRENT_ROW_L1, RECORD.EMPL_CHKLST_ITM, 1);
   End-For;

   If All(&FOUNDDOC) Then
      For &I = 1 To &SELECT_ROW
         CopyFields(1, RECORD.CHECKLIST_ITEM, &I, 2,
RECORD.EMPL_CHECKLIST, &CURRENT_ROW_L1, RECORD.EMPL_CHKLST_ITM, &I);
         If &I <> &SELECT_ROW Then
            InsertRow(RECORD.EMPL_CHECKLIST, &CURRENT_ROW_L1,
RECORD.EMPL_CHKLST_ITM, &I);
         End-If;
      End-For;
   End-If;
End-If;
```

The following example performs the same function as the previous code, only it uses the data buffer classes:

1. Flushes the rowset and hidden work scroll area (&RS1H).
2. Performs a Select statement on &RS1H based on the value of the CHECKLIST_CD field and the effective date.
3. Clears out the level two rowset (&RS2).
4. Copies like-named fields from &RS1H to &RS1.

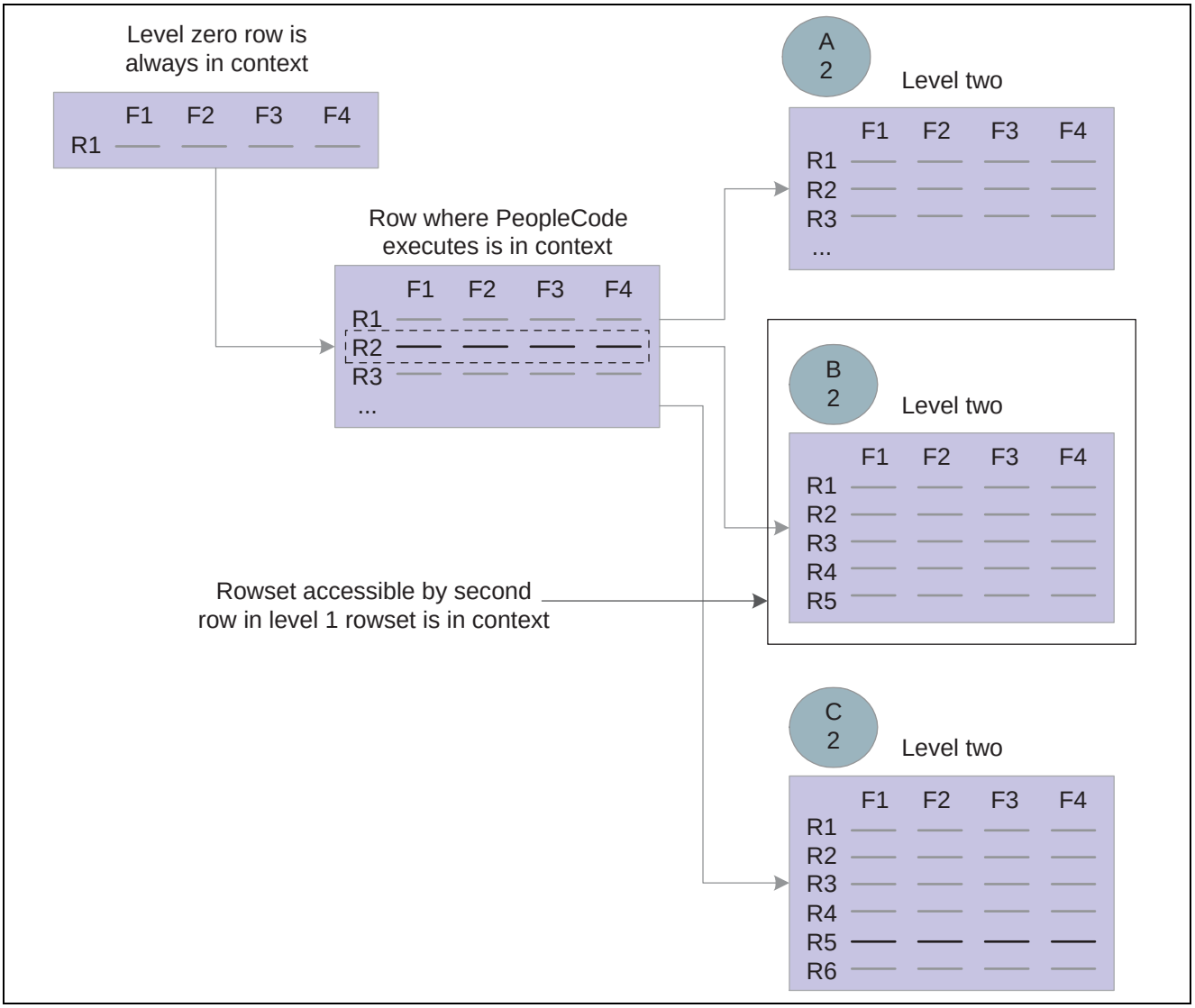
```
Local Rowset &RS0, &RS1, &RS2, &RS1H;

&RS0 = GetLevel0();
&RS1 = GetRowset();
&RS2 = GetRowset(SCROLL.EMPL_CHKLST_ITM);
&RS1H = &RS0.GetRow(1).GetRowset(SCROLL.CHECKLIST_ITEM);

&MYFIELD = CHECKLIST_CD;
```

```
If All(&MYFIELD) Then
   &RS1H.Flush();
   &RS1H.Select(RECORD.CHECKLIST_ITEM, "where Checklist_CD = :1
and EffDt = (Select Max(EffDt) from PS_CHECKLIST_ITEM
Where CheckList_CD = :2)", CHECKLIST_CD, CHECKLIST_CD);

   For &I = 1 To &RS2.ActiveRowCount
      &RS2.DeleteRow(1);
   End-For;

&FOUND = &RS1H.GetCurrEffRow().CHECKLIST_ITEM. CHKLST_ITEM_CD.Value;

   If All(&FOUND) Then
      For &I = 1 To &RS1H.ActiveRowCount
         &COPYFRMROW = &RS1H.getrow(&I);
         &COPYTROW = &RS2.getrow(&I);
     &COPYFRMROW.CHECKLIST_ITEM.CopyFieldsTo(&COPYTROW.EMPL_CHKLST_ITM);
         If &I <> &RS1H.ActiveRowCount Then
            &RS2.InsertRow(&I);
         End-If;
      End-For;
   End-If;
End-If;
```

Understanding Current Context

Most PeopleCode programs run in a current context. The current context determines which buffer fields can be contextually referenced from PeopleCode, and which row of data is the current row on each scroll level at the time a PeopleCode program is running.

The current context for the data buffer access classes is similar to the current context for accessing the component buffer, as shown in the following diagram:



Current context for rowsets

In this example, a PeopleCode program is running in a buffer field on the second row of the level one rowset. The following code returns a row object for the second row of the level one rowset, because that is the row that is the current context.

```
Local Row &ROW
&ROW = GetRow();
```

The following code returns the B2 level two rowset, because of the current context:

```
Local Rowset &ROWSET2
&ROWSET2 = &ROW.GetRowset(SCROLL.EMPL_CHKLST_ITM);
```

This code does not return either the C2 or the A2 rowsets. It returns only the rowset associated with the second row of the level one rowset.

Creating Records or Rowsets and Current Context

When you instantiate a record object using the CreateRecord function, you are only creating an area in the data buffers that has the same structure as the record definition. It does not contain any data. This record object does not have a parent rowset and is not associated with a row. It is a free-standing record object, and therefore is not considered part of the current context.

The same concept applies when you instantiate a rowset object using the CreateRowset function. You are only creating an area in the data buffers that has the same structure as the records or rowset the new rowset is based on. The rowset does not contain any data. This type of rowset does not have a parent rowset or row.

See Also

Chapter 4, "Referencing Data in the Component Buffer," Specifying Data with Contextual References, page 44

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

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

Accessing Secondary Component Buffer Data

When a secondary page is run, the data for its buffers is copied from the parent component to a buffer structure for the secondary page. This means there are two copies of this data. The data buffer classes give access to both of these copies of the data. Direct field references (recname.fieldname) always use the current context to determine which value to access. So, in general, when using a secondary page, make sure that references are based on the secondary page.

Instantiating Rowsets Using Non-Component Buffer Data

Both the application message and the file layout technologies represent hierarchical data, and use the rowset, row, record, and field hierarchy. Though you use different methods to instantiate a rowset object for this data, you still use the same rowset, row, record, and field methods and properties to manipulate the data. (Any exceptions are marked in the documentation.)

To instantiate a rowset for a message:

```
&MSG = CreateMessage(MESSAGE.EMPLOYEE_DATA);
&MYROWSET = &MSG.GetRowset();
```

To instantiate a rowset for a file layout:

```
&MYFILE = GetFile(&SOMENAME, "R");
&MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT);
&MYROWSET = &MYFILE.ReadRowset();
```

In an Application Engine program, the default state record is considered the primary record, and the main record in context. You can access the default state record using the following:

```
&STATERECORD = GetRecord();
```

If you have more than one state record associated with an Application Engine program, you can access them the same way you would access other, nonprimary data records, by specifying the record name. For example:

```
&ALTSTATE = GetRecord(RECORD.AE_STATE_ALT);
```

See Also

Chapter 8, “Using Methods and Built-In Functions,” Using Standalone Rowsets, page 152

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker, “Defining Message Channels and Messages”

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Engine, “Using Meta-SQL and PeopleCode”

CHAPTER 6

PeopleCode and the Component Processor

This chapter provides an overview of the Component Processor and discusses:

- Events outside the Component Processor flow.
- PeopleCode program triggers.
- Component Processor behavior.
- Processing sequences.
- PeopleSoft Pure Internet Architecture processing considerations.
- Deferred processing mode.
- PeopleCode events.
- PeopleCode execution in pages with multiple scroll areas.

Understanding the Component Processor

The Component Processor is the PeopleTools runtime engine that controls processing of an application from the time that a user requests a component from an application menu until the database is updated and processing of the component is complete.

Events Outside the Component Processor Flow

Messages also have events associated with them (OnRouteSend, OnRouteReceive, OnRequest and Subscription.) However, these events are only associated with the message definition, and are not associated with any page. Therefore, they are not considered part of the Component Processor flow.

An Application Engine program can have a PeopleCode program as an action. Though the right-hand drop-down list box on the PeopleCode Editor window displays the text *OnExecute*, the PeopleCode program really is not an event. Any PeopleCode contained in an Application Engine action is executed only when the action is executed.

A component interface can have user-defined methods associated with it. These methods are not part of any processor flow; they are called as needed by the program executing the component interface. In addition, the mobile events (OnConflict, OnSelect, OnValidate) are not part of the Component Processor flow, but part of the flow of mobile synchronization.

Security has a signon event during signon. This is actually PeopleCode programs on a record field that you've specified in setting up security.

Though application packages have a right-hand drop-down list box on the PeopleCode Editor window that displays the text *OnExecute*, this is not an event. Any PeopleCode contained in the application class is only executed when called explicitly in a PeopleCode program.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Component Interface Classes"

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Application Classes"

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Engine, "Creating Application Engine Programs," Specifying PeopleCode Actions

Enterprise PeopleTools 8.45 PeopleBook: Security Administration, "Understanding PeopleSoft Security"

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Mobile Agent, "Understanding PeopleSoft Mobile Agent"

PeopleCode Program Triggers

PeopleCode can be associated with a PeopleCode record field, a component record, and many other items. PeopleCode events are initiated at particular times, in particular sequences, during the course of the Component Processor's flow of execution. When an event is initiated, it triggers PeopleCode programs on specific objects.

The following items have events that are part of the Component Processor flow:

Items	Event Triggers
Menu items	Programs associated with the menu item
Component record fields	Programs on specific rows of data
Component records	Programs on specific rows of data
Components	Programs associated with the component
Pages	Programs associated with the page
Record fields	Programs on specific rows of data

Suppose a user changes the data in a page field, then presses TAB to move out of the field. This user action initiates the FieldEdit PeopleCode event. The FieldEdit event affects only the field and row where the change took place. If a FieldEdit PeopleCode program is associated with that record field, the program is executed once.

If you have two FieldEdit PeopleCode programs, one associated with the record field and a second associated with the component record field, both programs execute, but only on the specific field and row of data. The FieldEdit PeopleCode program associated with the first record field is initiated first, and then the FieldEdit PeopleCode program associated with the first component record field is initiated.

By contrast, suppose a user has opened a component for updating. As part of building the component, the Component Processor initiates the RowInit event. This event triggers RowInit PeopleCode programs on every record field on every row of data in the component. In a scroll area with multiple rows of data, every RowInit PeopleCode program is executed once for each row.

In addition, if you have RowInit PeopleCode associated with both the record field and the component record, both programs are executed against every record field on every row of data in the component. The RowInit PeopleCode program associated with the first record field is initiated first, and then the RowInit PeopleCode program associated with the first component record is initiated. If you set the value of a field with the record field RowInit PeopleCode, and then reset the field with the component record RowInit PeopleCode, the second value appears to the user.

When you develop with PeopleCode, you must consider when and where your programs are triggered during the Component Processor's flow of execution.

This section discusses how to:

- Access PeopleCode programs.
- Understand the execution order of events and PeopleCode.

See Also

Chapter 6, "PeopleCode and the Component Processor," Understanding Execution Order of Events and PeopleCode, page 84

Accessing PeopleCode Programs

Every PeopleCode program is associated with a PeopleCode event, and is often referred to by that name, such as RowInit PeopleCode or FieldChange PeopleCode. These programs are accessible from, and associated with, different items. The following table lists items and associated events.

Note. The SearchInit and SearchSave events (in the Component Record column of the table) are available only for the search record associated with a component.

Record Field Events	Component Record Field Events	Component Record Events	Component Events	Page Events	Menu Events
• FieldChange • FieldDefault • FieldEdit • FieldFormula • PrePopup • RowDelete • RowInit • RowInsert • RowSelect • SaveEdit • SavePostChg • SavePreChg • SearchInit • SearchSave • Workflow	• FieldChange • FieldDefault • FieldEdit • PrePopup	• RowDelete • RowInit • RowInsert • RowSelect • SaveEdit • SavePostChg • SavePreChg • SearchInit • SearchSave	• PostBuild • PreBuild • SavePostChg • SavePreChg • Workflow	Activate	ItemSelected

The following table lists types of PeopleCode programs and where to access them in PeopleSoft Application Designer.

PeopleCode Programs	Location in PeopleSoft Application Designer
Record field	Record definitions and page definitions
Component record field, component record, and component	Component definitions
Menu item	Menu definitions
Page field	Page definitions

Understanding Execution Order of Events and PeopleCode

In PeopleSoft, the component is the representation of a transaction. Therefore, any PeopleCode that is associated with a transaction should be in events associated with some level of the component. Code that should be executed every time a field is edited should be at the record field level. If you associate code with the correct transaction, you do not have to check for the component that is issuing it (such as surrounding your code with dozens of `If %Component =` statements). Records are more reusable, and code is more maintainable.

For example, if you have start and end dates for a course, you would always want to make sure that the end date was after the start date. Your program to check the dates would go on the SaveEdit at the record field level.

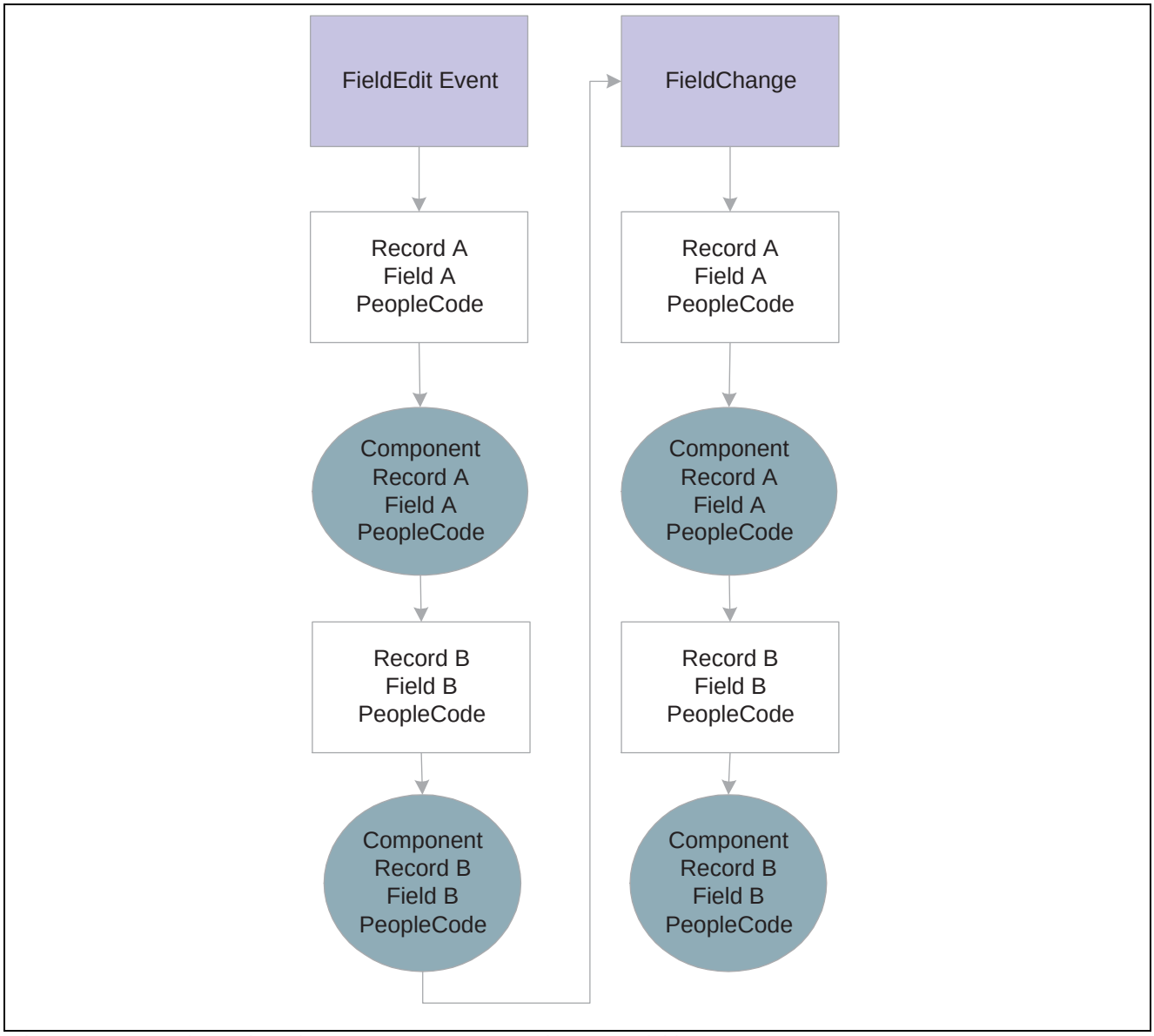
All similarly named component events are initiated after the like-named record event. The PeopleCode program associated with the record field event is initiated first, and then the PeopleCode program associated with the like-named component event is inititated. If you set the value of a field with the record field PeopleCode, and then reset the field with like-named component PeopleCode, the second value is displayed to the user.

Events After Field Changes

The following events occur after a user changes a field:

```
Record.recordA.fielda.FieldEdit -> Component.recordA.fielda.FieldEdit ->
Record.recordB.fieldb.FieldEdit -> Component.recordB.fieldb.FieldEdit ->
Record.recordA.fielda.FieldChange -> Component.recordA.fielda.FieldChange ->
Record.recordB.fieldb.FieldChange -> Component.recordB.fieldb.FieldChange ->
```

The following diagram shows the event flow:



Flow of events and PeopleCode programs after a user changes a field

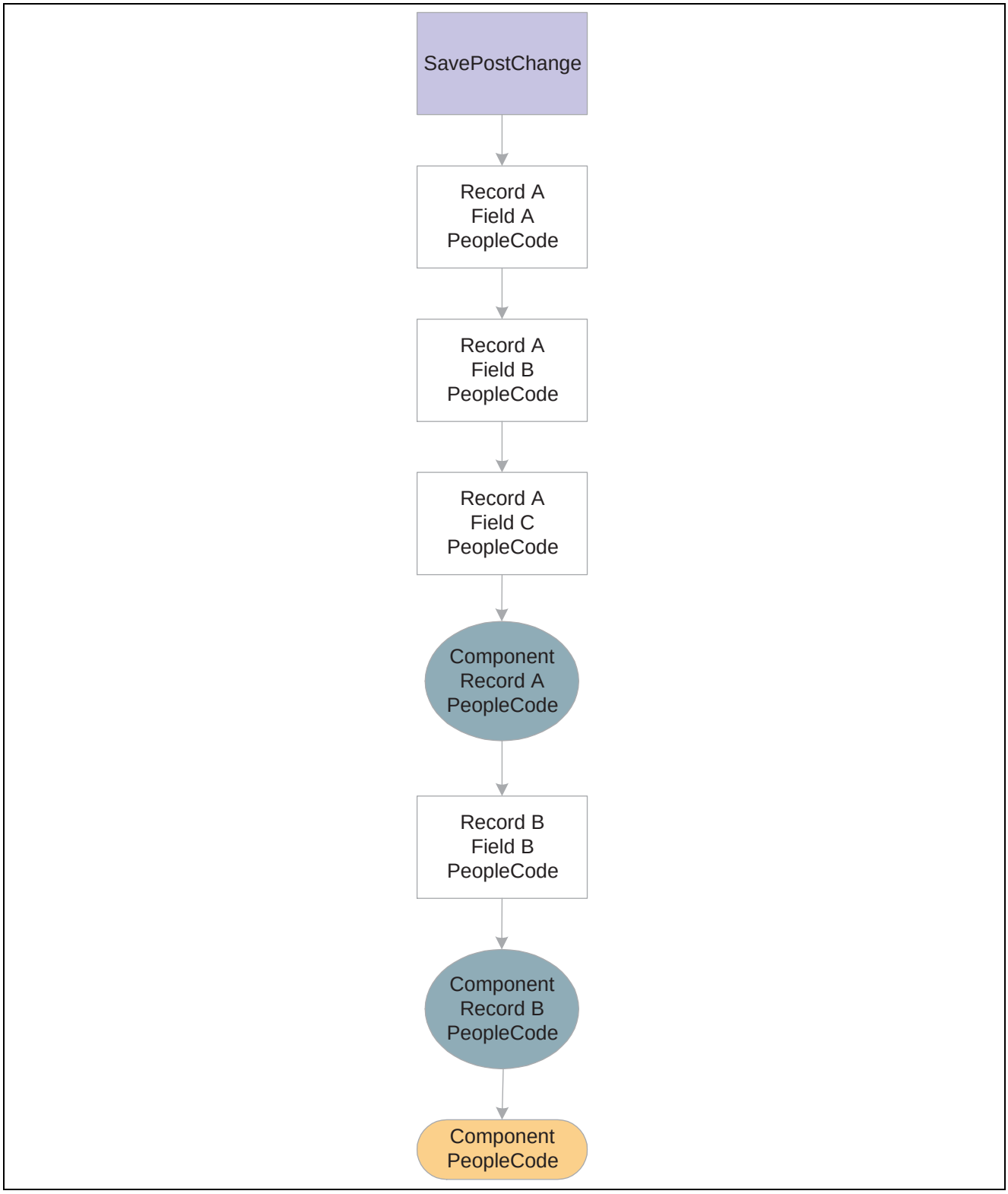
Events After User Saves

The following events occur after a user saves:

```
Record.recordA.fielda.SaveEdit ->
Record.recordA.fieldb.SaveEdit ->
Record.recordA.fieldc.SaveEdit ->
Component.recordA.SaveEdit

Record.recordB.fielda.SaveEdit ->
Record.recordB.fieldb.SaveEdit ->
Record.recordB.fieldc.SaveEdit ->
Component.recordB.SaveEdit

Record.recordA.fielda.SavePreChange ->
```

```
Record.recordA.fieldb.SavePreChange ->
Record.recordA.fieldc.SavePreChange ->
Component.recordA.SavePreChange

Record.recordB.fielda.SavePreChange ->
Record.recordB.fieldb.SavePreChange ->
Record.recordB.fieldc.SavePreChange ->
Component.recordB.SavePreChange

Record.recordA.fieldA.WorkFlow ->
Record.recordB.fieldB.WorkFlow ->
Record.reocrdC.fieldC.WorkFlow
Component.Workflow

Record.recordA.fielda.SavePostChange ->
Record.recordA.fieldb.SavePostChange ->
Record.recordA.fieldc.SavePostChange ->
Component.recordA.SavePostChange

Record.recordB.fielda.SavePostChange ->
Component.recordB.SavePostChange
Component.SavePostChange
```

The following diagram shows the event flow:



Flow of PeopleCode programs after SavePostChange event

Note. SaveEdit does not fire for deleted rows, but SavePreChange, Workflow, and SavePostChange do.

Component Processor Behavior

This section discusses:

- Component Processor behavior from page start to page display.
- Component Processor behavior following user actions in the component.

Note. Components that run in deferred mode behave differently.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Deferred Processing Mode, page 113

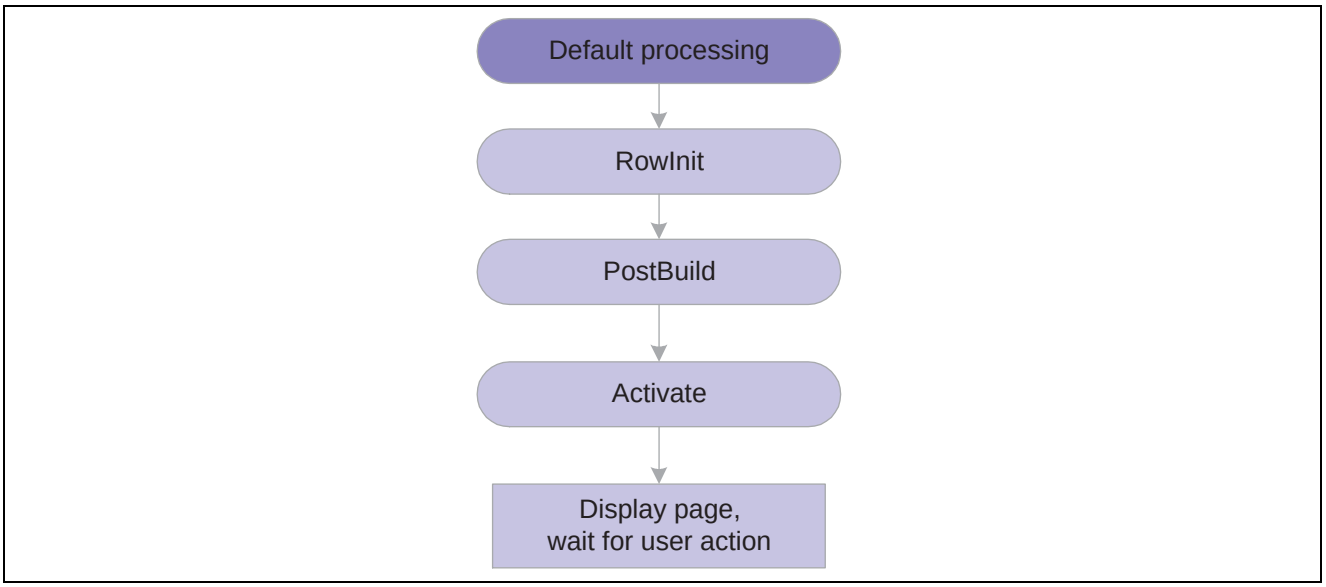
Chapter 6, “PeopleCode and the Component Processor,” Processing Sequences, page 91

Component Processor Behavior from Page Start to Page Display

Before a user selects a component, the system is in reset state, in which no component is displayed. The Component Processor’s flow of execution begins when a user selects a component from a PeopleSoft menu. The Component Processor then:

1. Performs search processing, in which it obtains and saves search key values for the component.
2. Retrieves from the database server any data needed to build the component.
3. Builds the component, creating buffers for the component data.
4. Performs any additional processing for the component or the page.
5. Displays the component and waits for user action.

The following flowchart shows the flow of execution at a high level:



Processing up to Page Display

Component Behavior Following User Actions in the Component

After a component is built and displayed, the Component Processor can respond to a number of possible user actions. The following table lists the user actions and briefly describes the resulting processing:

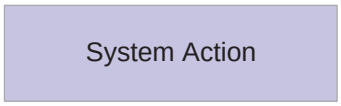
See Chapter 6, "PeopleCode and the Component Processor," Processing Sequences, page 91.

User Action	Description
Row Insert Processing	When a user requests a row insert, the Component Processor adds a row of data in the active scroll area, then displays the page again and waits for another action. See Chapter 6, "PeopleCode and the Component Processor," Row Insert Processing, page 106.
Row Delete Processing	When a user requests a row delete, the Component Processor flags the current row as deleted, then displays the page again and waits for another action. See Chapter 6, "PeopleCode and the Component Processor," Row Delete Processing, page 108.
Field Modification	If a user edits a page field, then leaves the field, the Component Processor performs standard edits (such as checking the data type and checking for values out of range). If the contents of the field do not pass the standard system edits, the Component Processor redisplays the page with an error or warning message and changes the field's color to the system color for field edit errors, usually red. Until the user corrects the error, the Component Processor does not let the user save changes or navigate to another field. If the contents of the field pass the standard system edits, the system redisplays the page and waits for further action. See Chapter 6, "PeopleCode and the Component Processor," Field Modification, page 103.
Prompts	If a user clicks the prompt icon next to a field, a list of values for the prompt field appears. If the user clicks Return To Search, or presses ALT+2, a search page appears, enabling the user to enter an alternate search key or partial value. If the end-user clicks the detail button next to a date field, a calendar appears.
Pop-up Menu Display	If a user clicks the pop-up icon next to a field, a pop-up menu appears. This can be a default pop-up menu or one that has been defined by the developer. If the user clicks the pop-up icon at the bottom of the page, the pop-up menu for the page appears. See Chapter 6, "PeopleCode and the Component Processor," Pop-Up Menu Display, page 110.

User Action	Description
ItemSelected Processing	A user can select an item from a pop-up menu to execute a command. See Chapter 6, "PeopleCode and the Component Processor," Selected Item Processing, page 110.
Push Button	A user can click a button to execute a command. See Chapter 6, "PeopleCode and the Component Processor," Buttons, page 109.
Save Processing	A user can direct the system to save a component by clicking Save or by pressing ALT+1. If any component data has been modified, the system also prompts the user to save a component when the Next or List button is clicked, or when a new action or component is selected. The Component Processor first validates the data in the component, and then updates the database with the changed component data. After the update, a SQL Commit command finalizes the changes. See Chapter 6, "PeopleCode and the Component Processor," Save Processing, page 110.

Processing Sequences

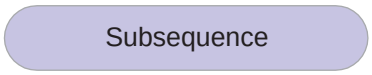
Actions and PeopleCode events can occur in various sequences within the Component Processor's flow of execution. Flow charts represent each sequence. In a flow chart, different shapes and colors represent different concepts.



Blue rectangles represent actions taken by the system.

Dark rhomboids represent branches (decision points) in the logic.

Dark ellipses represent PeopleCode events.



Light ellipses are subprocesses.

Most processing sequences correspond to high-level component processor behaviors. However, two important subsequences occur only in the context of a larger sequence. These subsequences are:

- Default processing, which occurs in a number of different contexts.
- Row select processing, which most commonly occurs as a part of component build in any of the update action modes.

  Row select processing also occurs when a ScrollSelect or related function is executed to load data into a scroll area.

See Chapter 6, "PeopleCode and the Component Processor," Component Processor Behavior, page 89; Chapter 6, "PeopleCode and the Component Processor," Default Processing, page 92 and Chapter 6, "PeopleCode and the Component Processor," Row Select Processing, page 101.

Note. Variations may occur in processing sequences, particularly when a PeopleCode function within a processing sequence initiates another processing sequence. For example, if a row of data is inserted or deleted programmatically during the component build sequence, this sets off a row insert or row delete sequence. Also note that components that run deferred mode behave differently.

See Chapter 6, "PeopleCode and the Component Processor," Deferred Processing Mode, page 113.

This section discusses:

- Default processing.
- Search processing in update mode.
- Search processing in add mode.
- Component build processing in update mode.
- Row select processing.
- Component build processing in add mode.
- Field modification.
- Row insert processing.
- Row delete processing.
- Buttons.
- Prompts.
- Pop-up menu display.
- Selected item processing.
- Save processing.

Default Processing

In default processing, any blank fields in the component are set to their default values. You can specify the default value either in the record field properties or in FieldDefault PeopleCode. If no default value is specified, the field is left blank.

Note. In PeopleSoft Pure Internet Architecture, if a user changes a field, but there is nothing to cause a trip to the server on that field, default processing and FieldFormula PeopleCode do not run. They only run when another event causes a trip to the server.

Default processing is relatively complex. The following two sections describe how default processing works on the level of the individual field, and how default processing works in the broader context of the component.

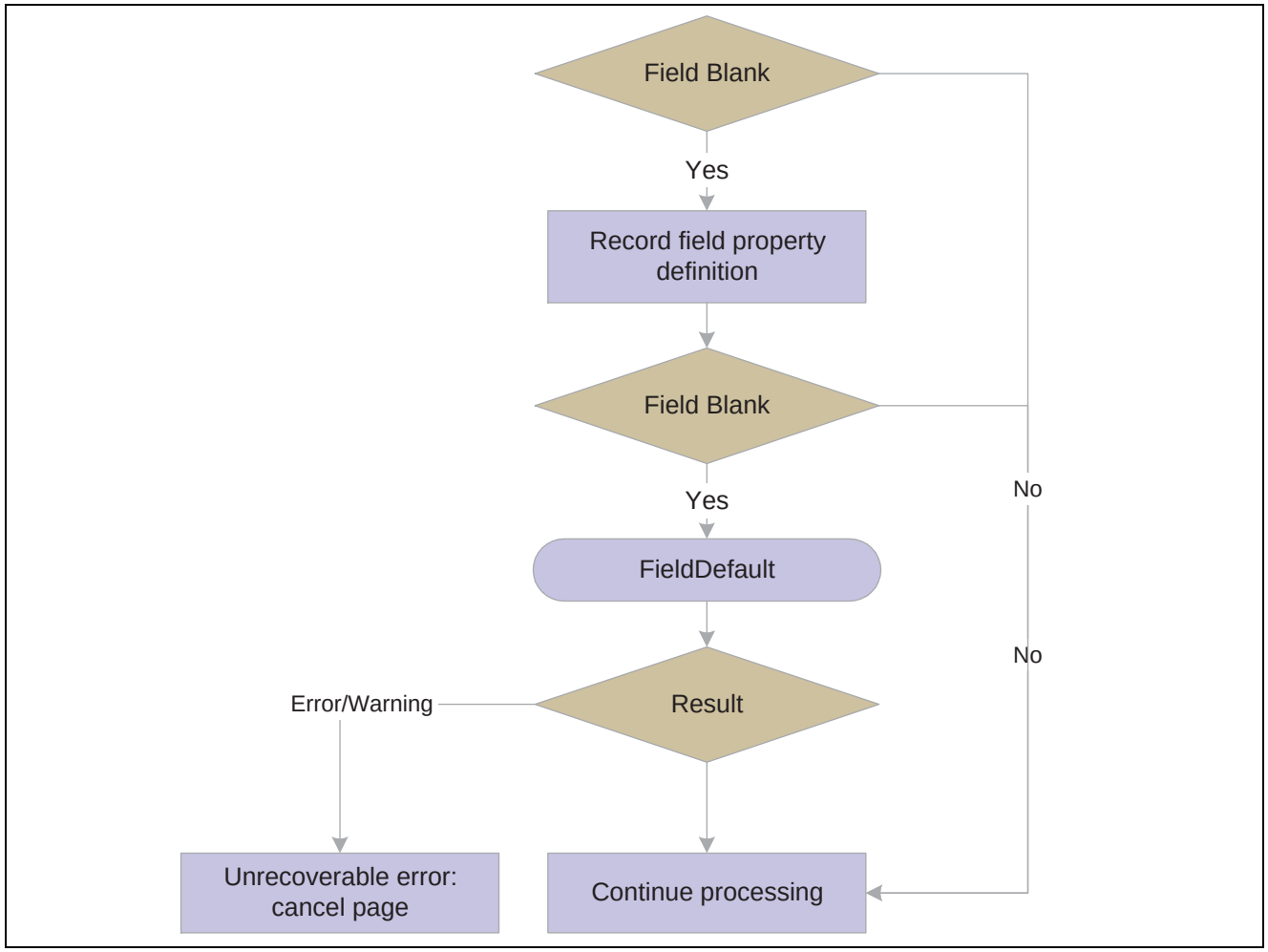
Field-Level Default Processing

During default processing, the Component Processor examines all fields in all rows of the component. On each field, it performs the following:

1. If the field is set to NULL (blank) for a character field, or set to 0 for a numeric field, the Component Processor sets the field to any default value specified in the record field properties for that field.
2. If no default value for the field is defined in the record field properties, then the Component Processor initiates the FieldDefault event, which triggers any FieldDefault PeopleCode associated with the record field or the component record field.
3. If an error or warning executes in any FieldDefault PeopleCode, a runtime error occurs.

Important! Avoid using error and warning statements in FieldDefault PeopleCode.

The following flowchart shows this logic:



Field-level default sequence flow

Default Processing on Component Level

Under normal circumstances, default processing in a component is relatively simple: each field on each row of data undergoes field-level default processing. For typical development tasks, this is all you need to be concerned with. However, the complete context of default processing is somewhat more complex.

See Chapter 6, "PeopleCode and the Component Processor," Default Processing, page 92.

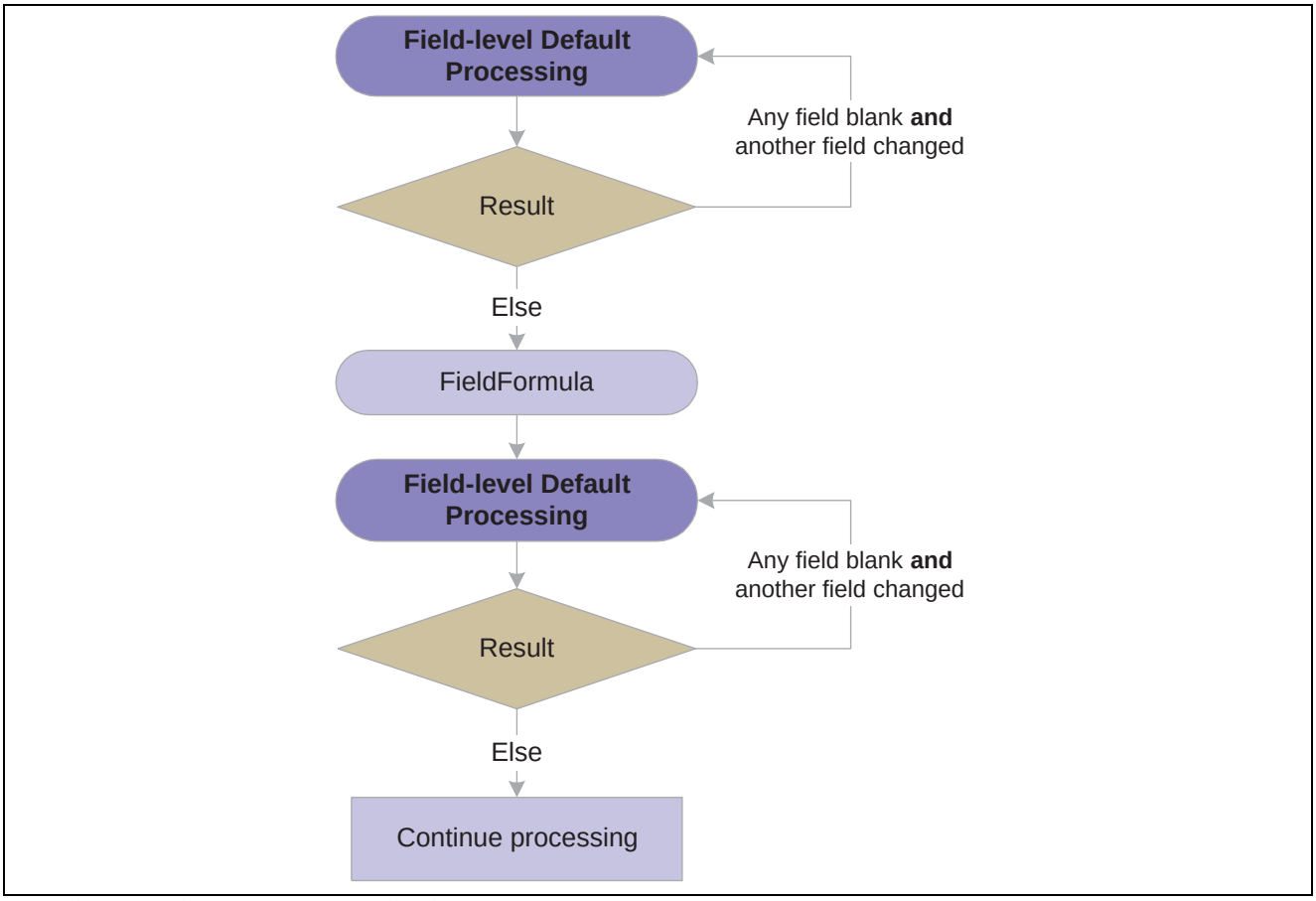
During component-level default processing, the Component Processor performs these tasks:

1. Field-level default processing is performed on all fields on all rows of data in the component.
2. If any field is still blank and any other field in the component has changed, field-level default processing may be repeated, in case a condition changed that causes default processing to now assign a value to something that was previously left blank.
3. The FieldFormula Event is initiated on all fields on all rows of data in the component.

   This PeopleCode event is often used for FUNCLIB_ (function library) record definitions to store shared functions, so normally no PeopleCode programs execute.
4. If the FieldFormula Event changed anything, field-level default processing is performed again, in case FieldFormula PeopleCode changed a field value to blank, or changed something that causes default processing to now assign a value to a field that was previously left blank.

   Because there should not be any FieldFormula PeopleCode, this is unlikely to affect the development process or performance.
5. If any field is still blank and any other field in the component has changed, field-level default processing is repeated.

The following flowchart shows this logic:



Default processing on component level

Search Processing in Update Modes

If a user selects any of the update action modes (Update, Update/Display All, or Correction), the Component Processor begins update mode search processing, which includes the following steps:

1. The SearchInit PeopleCode event is intiated, which triggers any SearchInit PeopleCode associated with the record field or the component search record, on the keys or alternate search keys in the component search record.

   This enables you to control the search page field values or the search page appearance programmatically, or to perform other processing prior to the appearance of the search page.

   Note. Set the search record for the component in the component properties.

   For example, the following program in SearchInit PeopleCode on the component search key record field EMPLID sets the search key page field to the user's employee ID, makes the page field unavailable for entry, and enables the user to modify the user's own data in the component:

```
EMPLID = %EmployeeId;
&Field = GetField(EMPLID).Enabled = False;
AllowEmplIdChg(true);
```

Note. Generally, the system search processing displays the search page. You can use the SearchInit event, and the SetSearchDialogBehavior function, to set the behavior of the search page before it is displayed. If SetSearchDialogBehavior is set to *Force display,* the dialog box is displayed even if all required keys have been provided. You can also set SetSearchDialogBehavior to *skip if possible.* In addition, you can force search processing to always occur by selecting Force Search Processing in the component definition properties in PeopleSoft Application Designer.

2. The search page and prompt list appear, in which the user can enter search keys or select an advanced search to enter alternate search keys.

   Note. Normally, the values in the search page are not set to default values. However, if the SearchDefault function was executed in SearchInit PeopleCode for any of the search key or alternate search fields, those fields in the dialog box are set to their system default values. No other default processing occurs (that is, the FieldDefault event is not initiated).

3. The user enters a value or partial value in the search page, and then clicks Search.
4. The SearchSave PeopleCode event is initiated, which triggers any SearchSave PeopleCode associated with the record field or the component search record, on the search keys or alternate search keys in the search record.

   This enables you to validate the user entry in the search page by testing the value in the search record field in PeopleCode and, if necessary, issuing an error or warning. If an error is executed in SearchSave, the user is sent back to the search page. If a warning is executed, the user can click OK to continue or click Cancel to return to the search page and enter new values.

   If partial values are entered, such that the Component Processor can select multiple rows, then the prompt list dialog box is filled, and the user can select a value. If key values from the search page are blank, or if the system cannot select any data based on the user entry in the search page, the system displays a message and redisplays the search page. If the values entered produce a unique value, the prompt list is not filled. Instead, the user is taken directly to the page.

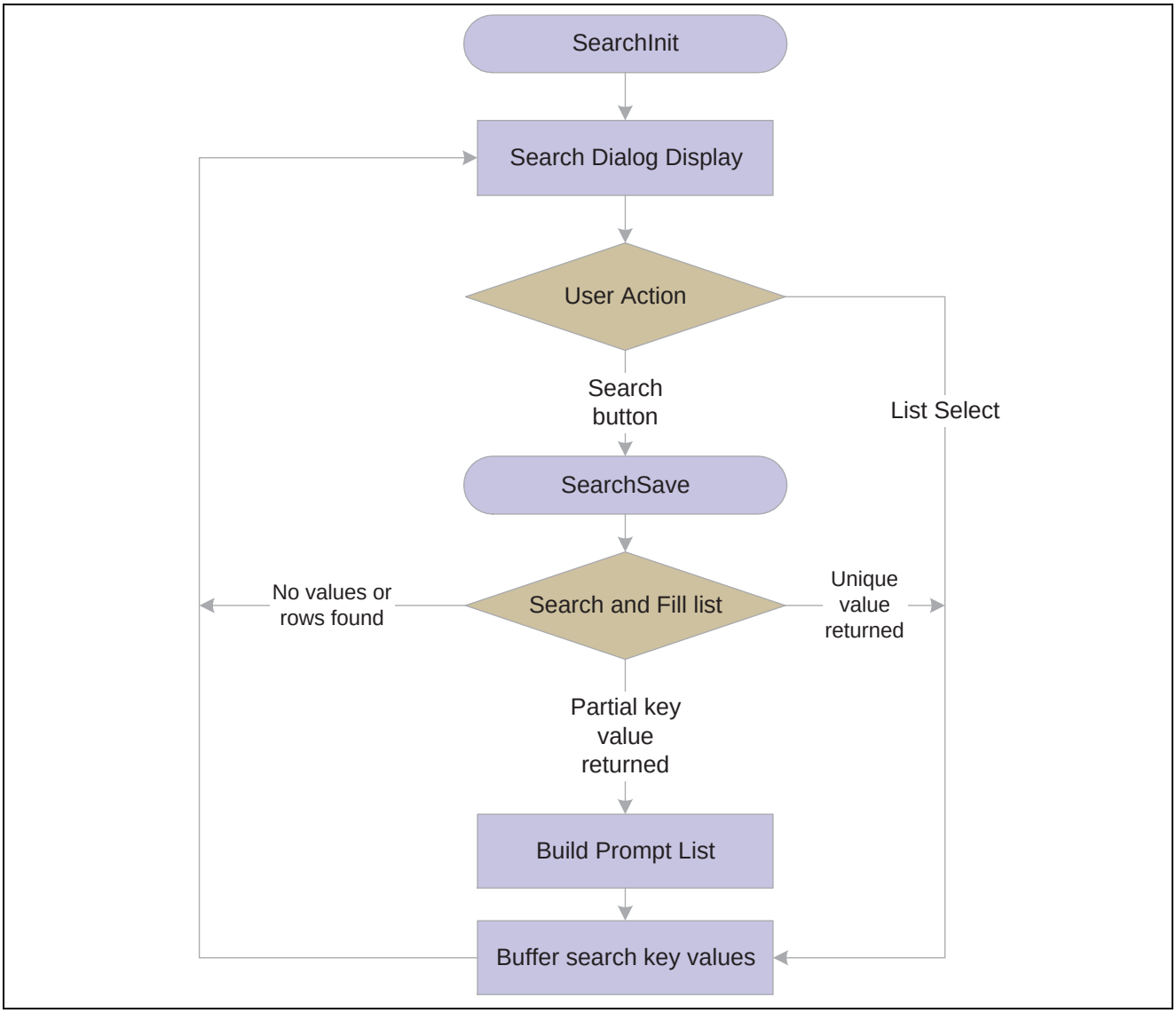
   Note. Normally, no system edits are applied when the user changes a field in the search page. However, if the SearchEdit property is executed for specific search page fields in SearchInit PeopleCode, the system edits are applied to those fields after the user changes a field and either leaves the field or clicks Search. In addition, the SearchEdit property can also be set in metadata for the record field definition.

   If the user entry in the field fails the system edits, the system displays a message, highlights the field in question, and returns theuser to the dialog box. The FieldEdit and SaveEdit PeopleCode events are not initiated. The SearchSave event is not initiated after values are selected from the search list. To validate data entered in the search page, use the Component PreBuild event.

5. The Component Processor buffers the search key values.

   If the user then opens another component while this component is active, the Component Processor uses the same search key values and bypasses the search page.

The following flowchart shows this logic. (It does not show the effects of executing the SearchDefault and SearchEdit Field class properties.)



Search processing logic in update mode

Note. You can use the IsSearchDialog built-in function to create PeopleCode that runs only during search processing. To create processes that run only in a specific action mode, use the %Mode system variable. This could be useful in code that is part of a library function and that is invoked in places other than from the search page. It could also be used in PeopleCode associated with a record field that appears in pages and in the search page.

See Also

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

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Field Class," SearchDefault

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "System Variables," %Mode

Search Processing in Add Modes

When a user opens a component in add or data-entry modes, the following actions occur:

1. The Component Processor runs default processing on the high-level keys that appear in the Add or Data Entry dialog box.
2. The Component Processor initiates the RowInit event, which triggers any RowInit PeopleCode associated with the record field or the component record, on the Add or Data Entry dialog box fields.
3. The Component Processor initiates the SearchInit event on dialog fields, which triggers any SearchInit PeopleCode associated with the record field or the component search record.

   This enables you to execute PeopleCode programs before the dialog box appears.
4. The Component Processor displays the Add or Data Entry dialog box.
5. If the user changes a dialog box field, and then leaves the field or clicks OK, the following actions occur:
   - In add mode only, a field modification processing sequence occurs.

     See Chapter 6, "PeopleCode and the Component Processor," Field Modification, page 103.
   - Default processing is run on the Add or Data Entry dialog box fields.

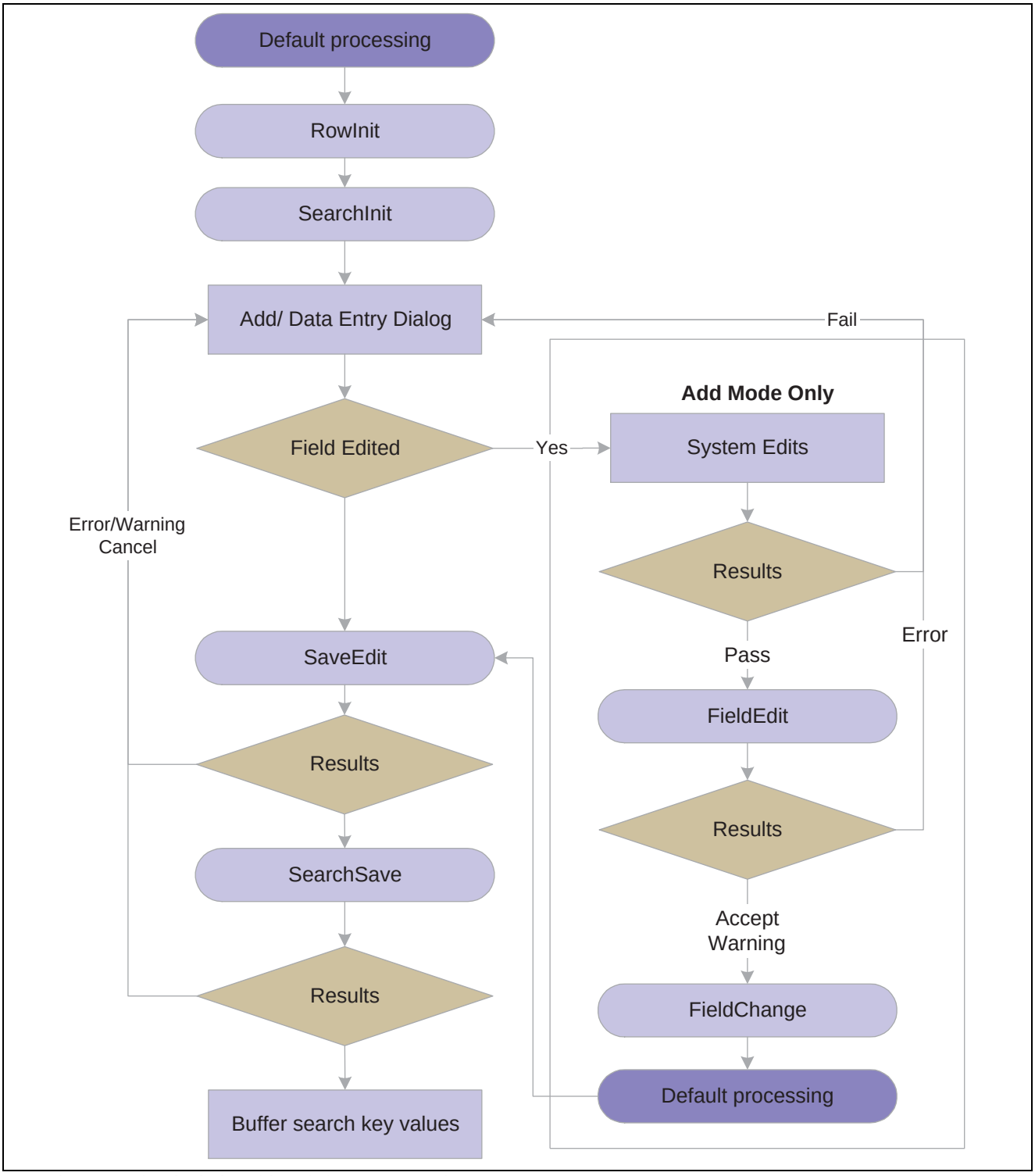
     Normally this does not have any effect, because the fields have a value.
6. When the user clicks OK in the dialog box, the SaveEdit event is initiated, which triggers any PeopleCode associated with the record field or the component record.
7. The Component Processor initiates the SearchSave event, which triggers any SearchSave PeopleCode associated with the record field or the component search record.

   This enables you to validate user entry in the dialog box. If an error is executed in SearchSave, the user is sent back to the Add or Data Entry dialog box. If a warning is executed, the user can click OK to continue or click Cancel to return to the dialog box and enter new values.
8. The Component Processor buffers the search key values and continues processing.

Note. If you compare the following diagram with search processing in update modes, notice that the add modes are considerably more complex and involve more PeopleCode events. However, in practice, PeopleCode development is similar in both cases. PeopleCode that runs before the dialog box appears (for example, to control dialog box appearance or set values in the dialog box fields) generally is placed in the SearchInit event; PeopleCode that validates user entry in the dialog box is placed in the SearchSave event.

See Chapter 6, "PeopleCode and the Component Processor," Search Processing in Update Modes, page 95.

The following flowchart shows this logic.



Search processing logic in add and data-entry modes

Note. You can use the IsSearchDialog function to create PeopleCode that runs only during search processing. To create processes that run only in a specific action mode, use the %Mode system variable. This could be useful in code that is part of a library function and that is invoked in places other than from the search page. It could also be used in PeopleCode associated with a record field that appears in pages and in the search page.

See Also

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

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "System Variables," %Mode

Component Build Processing in Update Modes

After the Component Processor has saved the search keys values for the component, it uses the search key values to select rows of data from the database server using a SQL Select statement. After the rows are retrieved, the Component Processor performs these actions:

1. Performs row select processing, in which rows of data that have already been selected from the database server can be filtered before they are added to the component buffer.

   See Chapter 6, "PeopleCode and the Component Processor," Row Select Processing, page 101.

2. Initiates the PreBuild event, which triggers any PreBuild PeopleCode associated with the component record, enabling you to set global or component scope variables that can be used later by PeopleCode located in other events.

   The PreBuild event is also used to validate data entered in the search page, after a prompt list is displayed.

   Note. If a PreBuild PeopleCode program issues an error or warning, the user is returned to the search page. If there is no search page, that is, the search record has no keys, a blank component page appears.

3. Performs default processing on all the rows and fields in the component.

   See Chapter 6, "PeopleCode and the Component Processor," Default Processing, page 92.

4. Initiates the RowInit event, which triggers any RowInit PeopleCode associated with the record field or the component record.

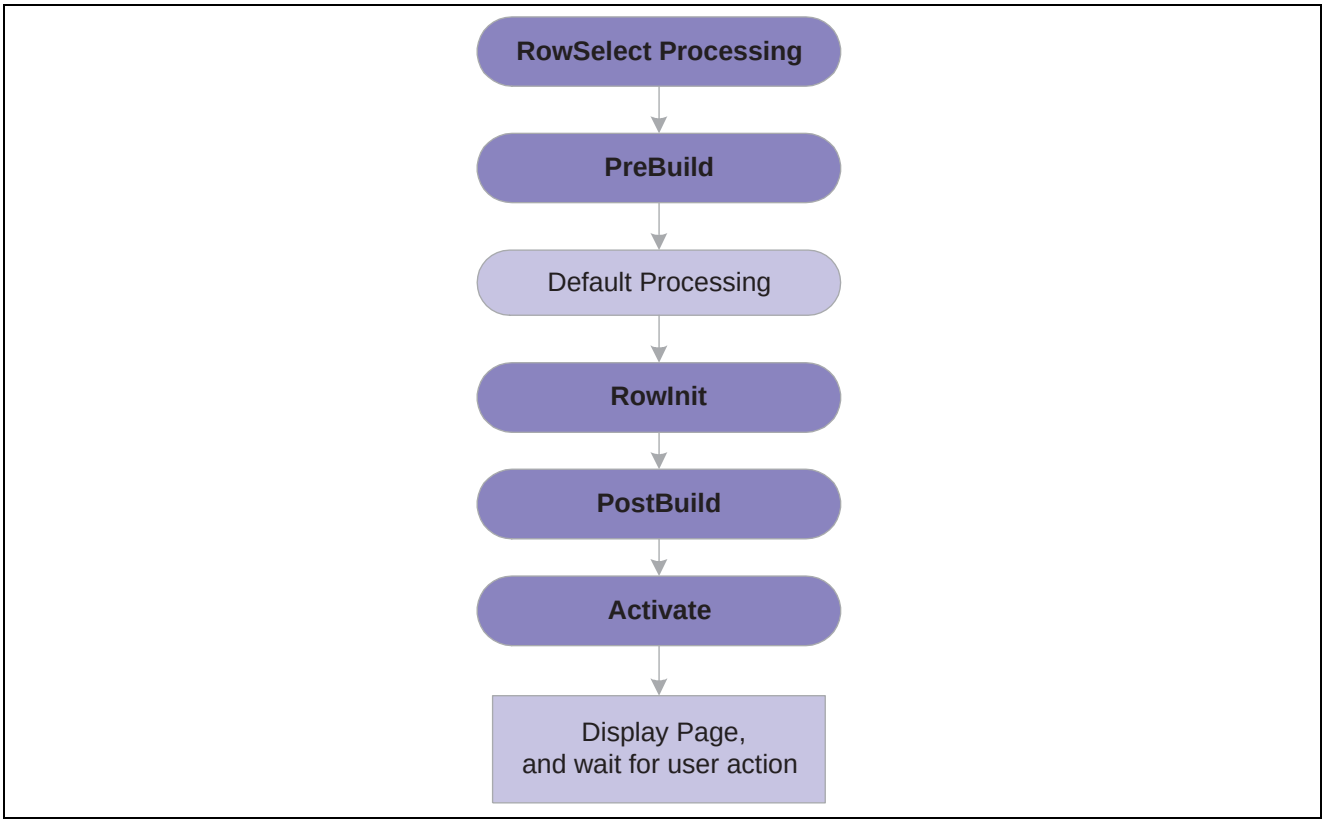
   The RowInit event enables you to programmatically initialize the values of non-blank fields in the component.

5. Initiates the PostBuild event, which triggers any PostBuild PeopleCode associated with the component record, enabling you to set global or component scope variables that can be used later by PeopleCode located in other events.

6. Initiates the Activate event, which triggers any Activate PeopleCode associated with the page about to be displayed, enabling you to programmatically control the display of that page.

7. Displays the component and waits for end-user action.

The following flowchart shows this logic.



Component build processing in update modes

Row Select Processing

Row select processing enables PeopleCode to filter out rows of data after they have been retrieved from the database server and before they are copied to the component buffers. Row select processing uses a SQL Select statement .

Row select processing is a subprocess of component build processing in add modes. It also occurs after a ScrollSelect or related function is executed.

See Chapter 6, "PeopleCode and the Component Processor," Component Build Processing in Add Modes, page 102.

Note. Instead of using row select processing, it is more efficient to filter out the rows using a search view, an effective-dated record, the Select method, or ScrollSelect or a related function, before the rows are sent to the browser.

In row select processing, the following actions occur:

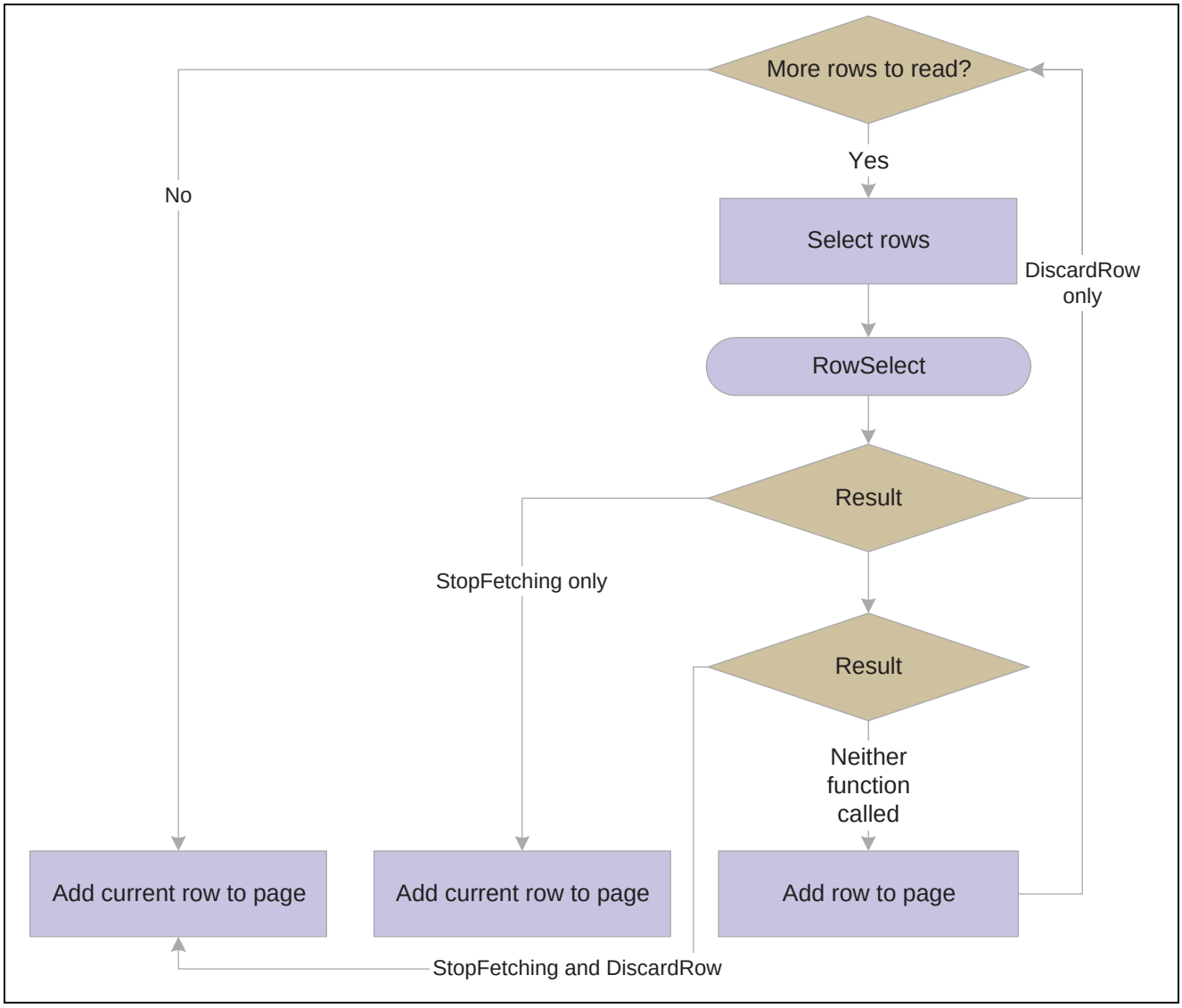
1. The Component Processor checks for more rows to add to the component.
2. The Component Processor initiates the RowSelect event, which triggers any RowSelect PeopleCode associated with the record field or component record.

   This enables PeopleCode to filter rows using the StopFetching and DiscardRow functions. StopFetching causes the system to add the current row to the component, and then to stop adding rows to the component. DiscardRow filters out a current row, and then continues the row select process.
3. If neither the StopFetching nor DiscardRow function is called, the Component Processor adds the rows to the page and checks for the next row.

The process continues until there are no more rows to add to the component buffers. If both StopFetching and DiscardRow are called, the current row is not added to the page, and no more rows are added to the page.

Note. In RowSelect PeopleCode, you can refer only to record fields on the record that is currently being processed, because the buffers are in the process of being populated. This means that the data might not be present.

The following flowchart shows this logic:



RowSelect processing logic

See Also

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

Component Build Processing in Add Modes

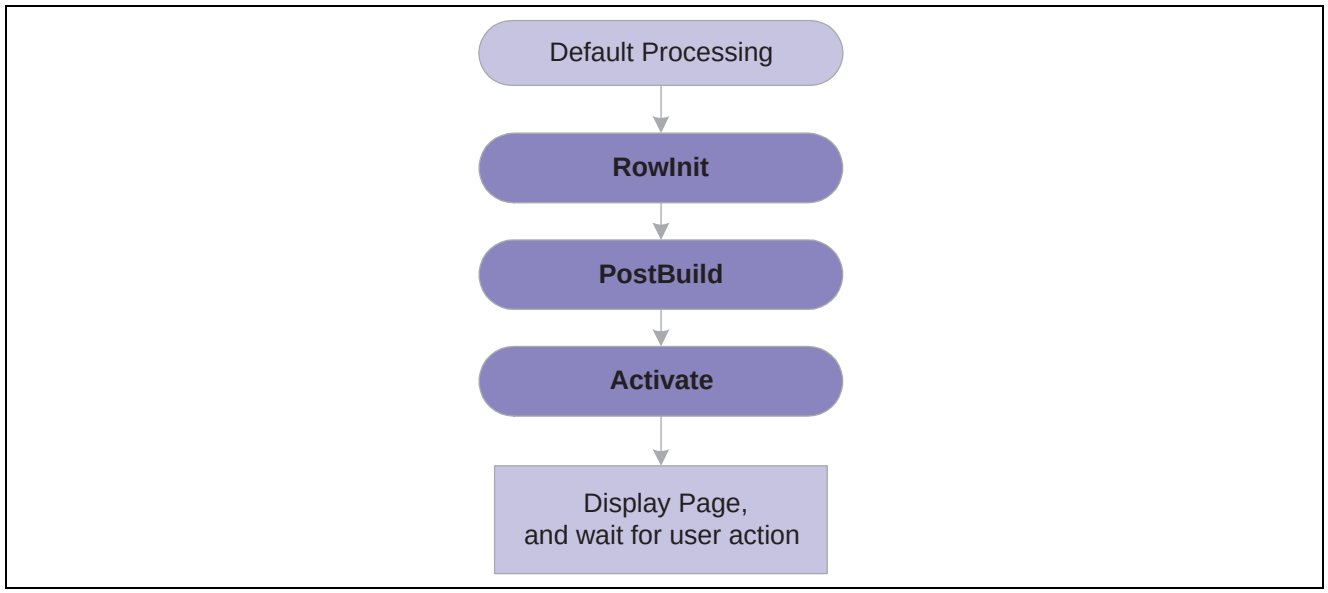
After search processing in add or data-entry modes, the Component Processor:

1. Initiates the PreBuild event.
2. Runs default processing on all page fields.

   This enables you to set default fields programmatically using FieldDefault PeopleCode.
3. Initiates the RowInit event on all fields in the component, which triggers any RowInit PeopleCode associated with the record field or component record.

   This enables you to initialize the state of page controls, using RowInit PeopleCode, before the controls are displayed. (RowInit enables you to set the values of non-blank fields programmatically, whereas default processing is used to set blank fields to their default values.)
4. Initiates the PostBuild event, which triggers any PostBuild PeopleCode associated with the component record, enabling you to set global or component scope variables that can be used later by PeopleCode located in other events.
5. Initiates the Activate event, which triggers any Activate PeopleCode associated with the page about to be displayed, enabling you to programmatically control the display of that page.
6. Displays a new component, using the search keys obtained from the Add or Data Entry dialog box, with other fields set to their default values.

The following flowchart shows the logic:



Logic of component build processing in add modes

Field Modification

The field modification processing sequence occurs after a user does any of the following:

- Changes the contents of a field, and then leaves the field.
- Changes the state of a radio button or check box.
- Clicks a command button.

In this sequence, the following actions occur:

1. The Component Processor performs standard system edits.

To reduce trips to the server, some processing must be done locally on the machine where the browser is located, while some is performed on the server.

Standard system edits can be done either on the browser, utilizing local JavaScript code, or on the application server. The following table outlines where these system edits are done.

System Edits	Location of Execution
Checking data type	Browser
Formatting	Application server or browser
Updating current or history record	Application server
Effective date	Application server
Effective date or sequence	Application server
New effective date in range	Application server
Duplicate key	Application server
Current level is not effective-dated but one of its child scroll areas is	Application server
Required field	Browser
Date range	Browser
Prompt table	Application server
Translate table	Browser
Yes/no table	Depends on the field type. Browser if the field is a check box. Application server if the field is an edit box and the values are *Y* or *N*.

Note. Default processing for the field can be done on the browser only if the default value is specified as a constant in the record field properties. If the field contains a default, these defaults occur only upon component initialization. Then, if a user replaces a default value with a blank, the field is not reinitialized. The required fields check is not performed on derived work fields when you press TAB to move out of a field.

If the data fails the system edits, the Component Processor displays an error message and highlights the field in the system color for errors (usually red).

2. If the field passes the system edits, Component Processor initiates the FieldEdit PeopleCode event, which triggers any FieldEdit PeopleCode associated with the record field or the component record field.

   This enables you to perform additional data validation in PeopleCode. If an Error statement is called in any FieldEdit PeopleCode, the Component Processor treats the error as it does a system edit failure; a message is displayed, and the field is highlighted. If a Warning statement is executed in any FieldEdit PeopleCode, a warning message appears, alerting the user to a possible problem, but the system accepts the change to the field.

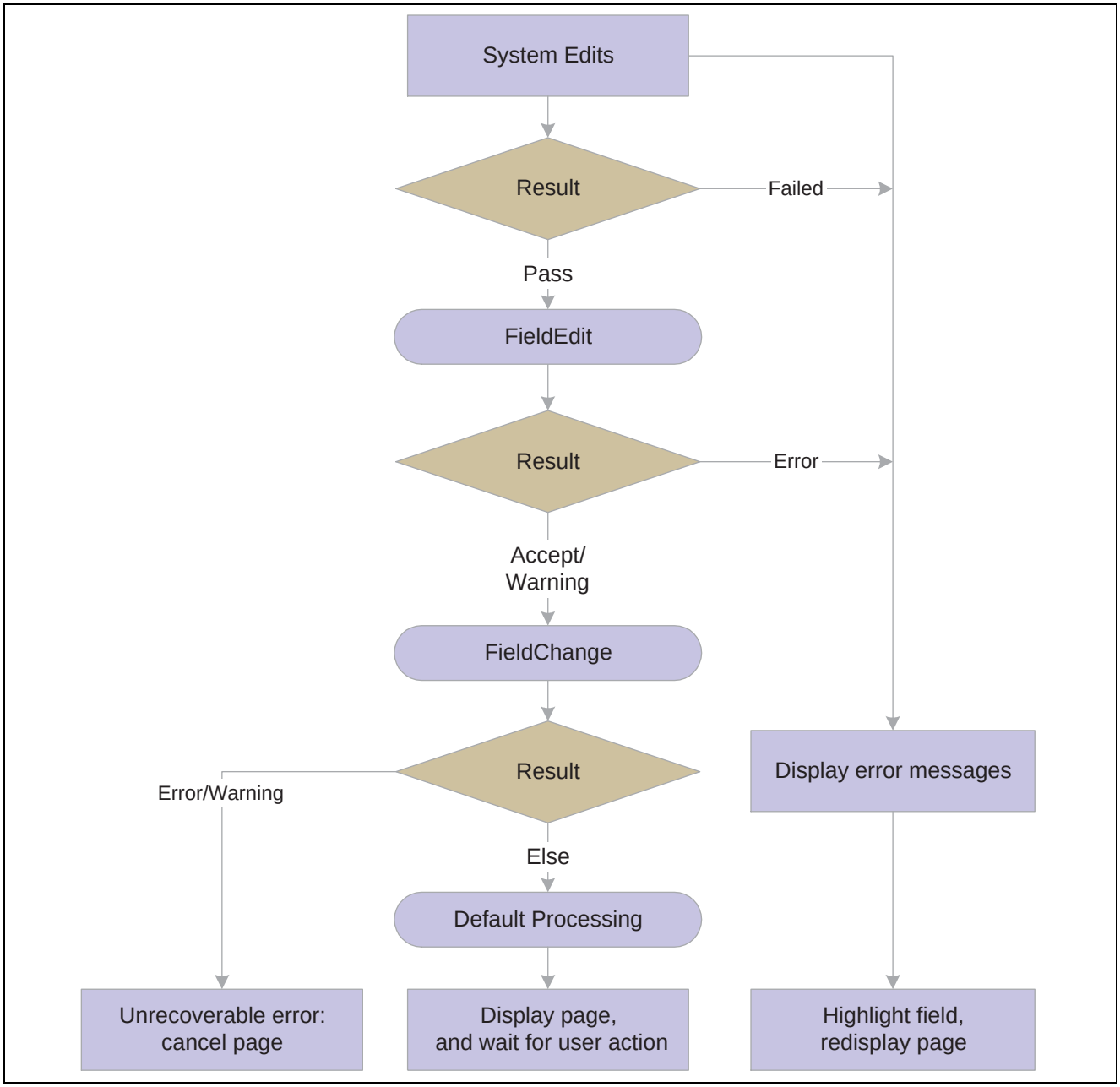
3. If the field change is accepted, the Component Processor writes the change to the component buffer, then initiates the FieldChange event, which triggers any FieldChange PeopleCode associated with the record field or the component record field.

   This event enables you to add processes other than validation initiated by the changed field value, such as changes to page appearance or recalculation of values in other page fields. An Error or Warning statement in any FieldChange PeopleCode causes a runtime error.

   Important! Do not use Error or Warning statements in FieldChange PeopleCode. All data validation should be performed in FieldEdit PeopleCode.

   After FieldChange processing, Component Processor runs default processing on all page fields, then redisplays the page. If the user has changed the field value to a blank, or if SetDefault or a related function is executed, and the changed field has a default value specified in the record field definition or any FieldDefault PeopleCode, the field is reinitialized to the default value.

The following flowchart shows this logic:



Logic of field modification processing

Row Insert Processing

Row insert processing occurs when:

- A user requests a row insert in a scroll area by pressing ALT+7, by clicking the Insert Row button, or by clicking the New button.
- A PeopleCode RowInsert function or a InsertRow method requests a row insert.

In either case, the Component Processor performs these actions:

1. Inserts a new row of data into the active scroll area.

If the scroll area has a dependent scroll area, the system inserts a single new row into the blank scroll area, and the system continues until it reaches the lowest-level scroll area.

2. Initiates the RowInsert PeopleCode event, which triggers any RowInsert PeopleCode associated with the record field or the component record.

   This event processes fields only on the inserted row and any dependent rows that were inserted on lower-level scroll areas.

3. Runs default processing on all component fields.

   Normally this affects only the inserted row fields and fields on dependent rows, because other rows already have undergone default processing.

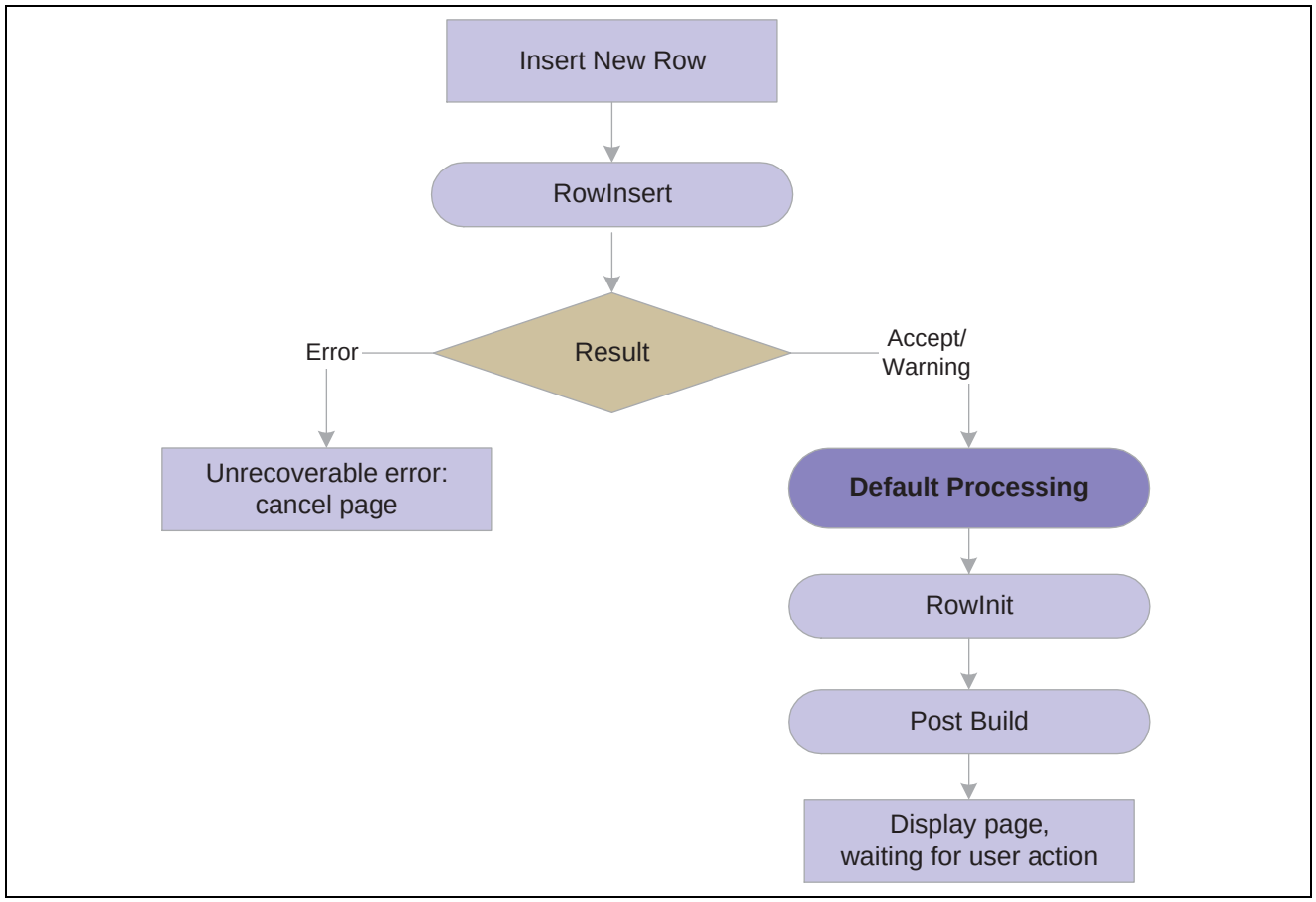
4. Initiates the RowInit PeopleCode event, which triggers any RowInit PeopleCode associated with the record field or the component record.

   This event affects fields only on the inserted row and any dependent rows that were inserted.

5. Redisplays the page and waits for user action.

Important! Do not use Error or Warning statements in RowInsert PeopleCode. All data validation should be performed in FieldEdit or SaveEdit PeopleCode.

The following flowchart shows this logic:



Logic of row insert processing

Note. If none of the data fields in the new row are changed after the row has been inserted (either programmatically or by the user), the new row is not inserted into the database when the page is saved.

Row Delete Processing

Row delete processing occurs when:

- A user requests a row delete in a scroll area by pressing ALT+8, by clicking the Delete Row button, or by clicking the Delete button.
- A PeopleCode RowDelete function or a DeleteRow method requests a row delete.

In either case, these actions occur:

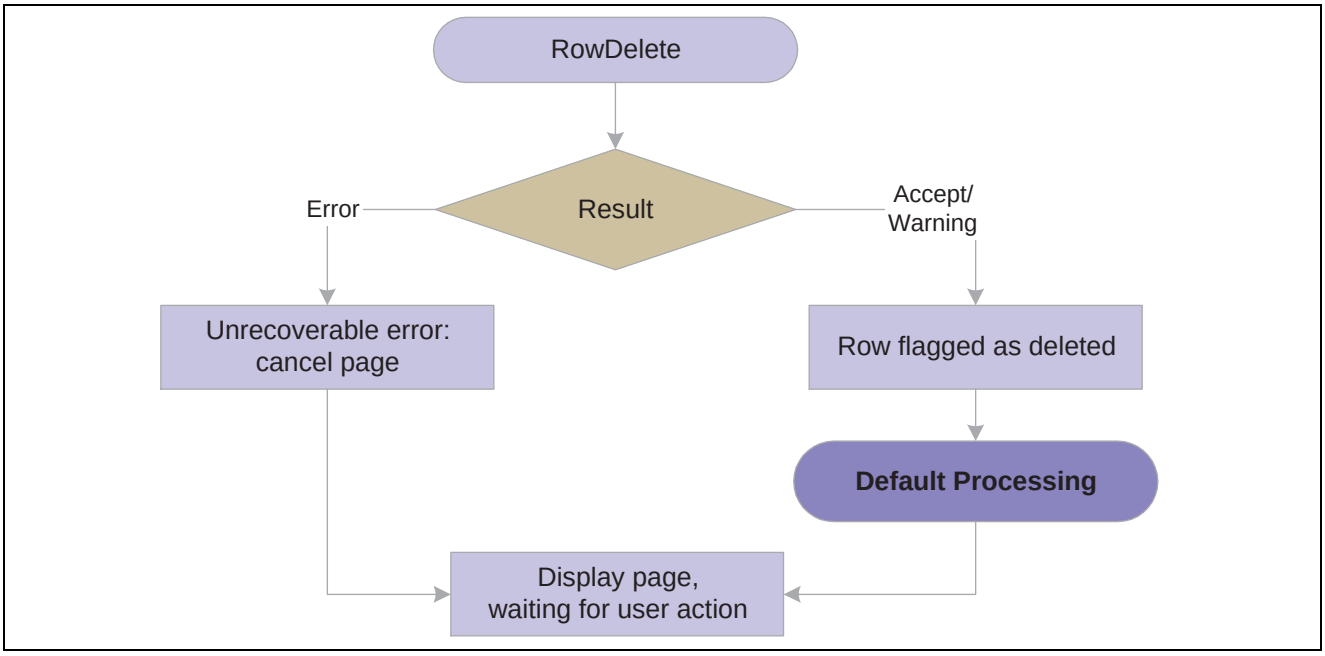
1. The Component Processor initiates the RowDelete PeopleCode event, which triggers RowDelete PeopleCode associated with the record field or the component record.

   This event processes fields on the deleted row and any dependent child scroll areas. RowDelete PeopleCode enables you to check for conditions and control whether a user can delete the row. An Error statement displays a message and prevents the user from deleting the row. A Warning statement displays a message alerting the user about possible consequences of the deletion, but permits deletion of the row.
2. If the deletion is rejected, the page is redisplayed after the error message.
3. If the deletion is accepted, the row, and any child scroll areas dependent on the row, are flagged as deleted.

   The row no longer appears in the page, but it is not physically deleted from the buffer and can be accessed by PeopleCode all the way through the SavePostChange event (note, however, that SaveEdit PeopleCode is not run on deleted rows).
4. The Component Processor runs default processing on all component fields.
5. The Component Processor redisplays the page and waits for a user action

   Note. PeopleCode programs are triggered on rows flagged as deleted in SavePreChange and SavePostChange PeopleCode. Use the IsDeleted row class property to test whether a row has been flagged as deleted. You can also access rows flagged as deleted by looping through the rows of a scroll area using a For loop delimited by the value returned by the RowCount rowset property.

The following flowchart shows this logic:



Logic of row delete processing

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Row Class," IsDeleted

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Rowset Class," RowCount

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

Buttons

When a user presses a button, this initiates the same processing as changing a field. Typically, PeopleCode programs started by button are placed in the FieldChange event.

See Also

Chapter 6, "PeopleCode and the Component Processor," Field Modification, page 103

Prompts

No PeopleCode event is initiated as a result of prompts, returning to the search page or displaying a calendar. This process is controlled automatically by the system.

Note. When the value of a field is changed using a prompt, the standard field modification processing occurs.

See Also

Chapter 6, "PeopleCode and the Component Processor," Field Modification, page 103

Pop-Up Menu Display

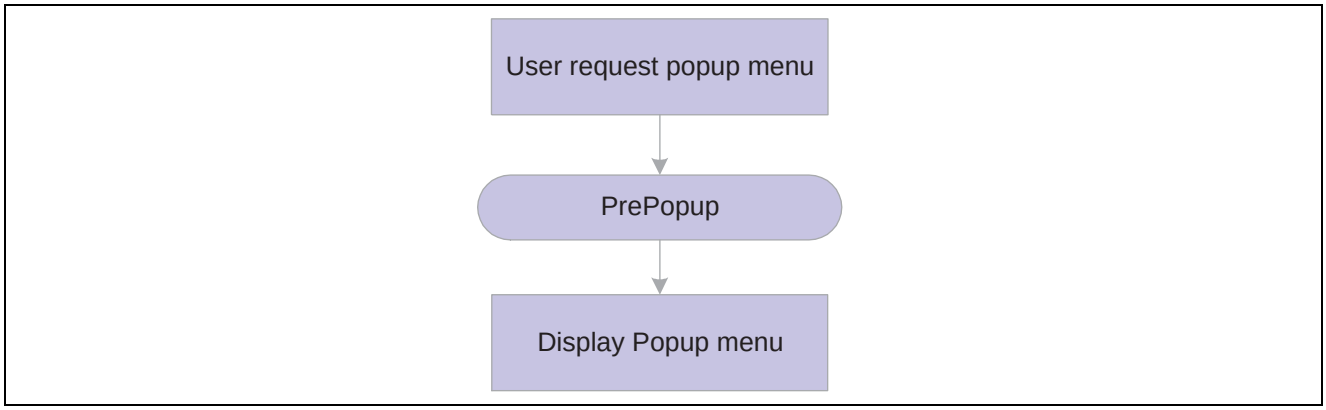
To display a pop-up menu, a user can click the pop-up button, either next to a field or at the bottom of a page (if the page has a pop-up menu associated with it.) The user can open a standard pop-up menu on a page field if no pop-up menu has been defined by an application developer for that page field.

The PrePopup PeopleCode event initiates only if the user opens a pop-up menu defined by an application developer on a page field. It does not initiate for a pop-up menu attached to the page background.

The PrePopup PeopleCode event enables you to disable, check, or hide menu items in the pop-up menu.

PrePopup PeopleCode menu item operations (such as HideMenuItem, EnableMenuItem, and so on) work with pop-up menus attached to a grid, not a field in a grid, only if the PrePopup PeopleCode meant to operate on that pop-up menu resides in the record field that is attached to the first column in the grid. It does not matter if the first field is visible or hidden.

The following flowchart shows this logic:

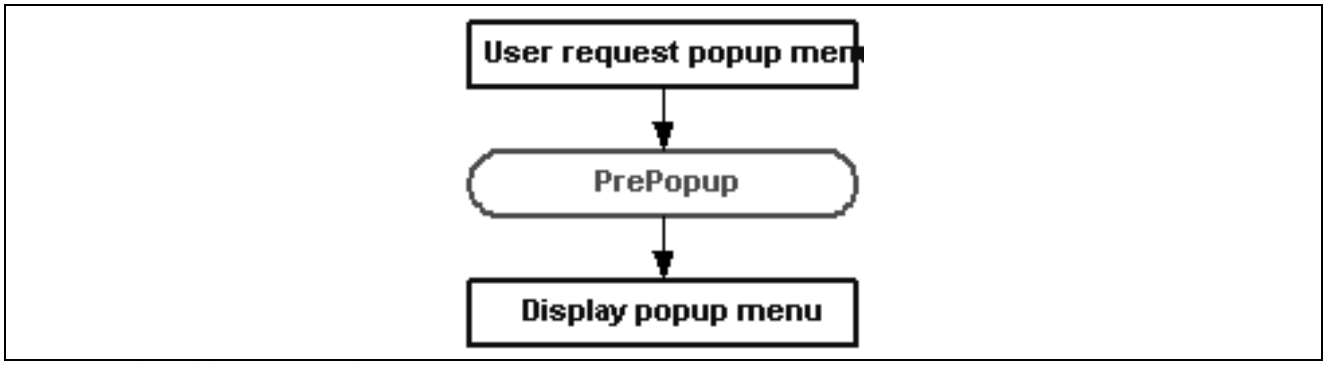


Logic of PrePopup even processing

Selected Item Processing

Selected item processing occurs when a user selects a menu item from a pop-up menu. This initiates the ItemSelected PeopleCode event, which is a menu PeopleCode event.

The following flowchart shows this logic:



Logic of selected item processing

Save Processing

A user can direct the system to save a component by clicking Save or by pressing ALT+1.

An application can prompt the user to save a component when the Next or List button is clicked, or when a new action or component is selected. If the user clicks Save after being prompted, save processing begins.

The following actions occur in save processing:

1. The Component Processor initiates the SaveEdit PeopleCode event, which triggers any SaveEdit PeopleCode associated with a record field or a component record.

   This enables you to cross-validate page fields before saving, checking consistency among the page field values. An Error statement in SaveEdit PeopleCode displays a message and then redisplays the page, stopping the save. A Warning statement enables the user to cancel save processing by clicking Cancel, or to continue with save processing by clicking OK.

2. The Component Processor initiates the SavePreChange event, which triggers any SavePreChange PeopleCode associated with a record field, a component record, or a component.

   SavePreChange PeopleCode enables you to process data after validation and before the database is updated.

3. The Component Processor initiates the Workflow event, which triggers any Workflow PeopleCode associated with a record field or a component.

   Workflow PeopleCode should be used only for workflow-related processing (TriggerBusinessEvent and related functions).

4. The Component Processor updates the database with the changed component data, performing any necessary SQL Insert, Update, and Delete statements.

5. The Component Processor initiates the SavePostChange PeopleCode event, which triggers any SavePostChange PeopleCode associated with a record field, a component record, or a component.

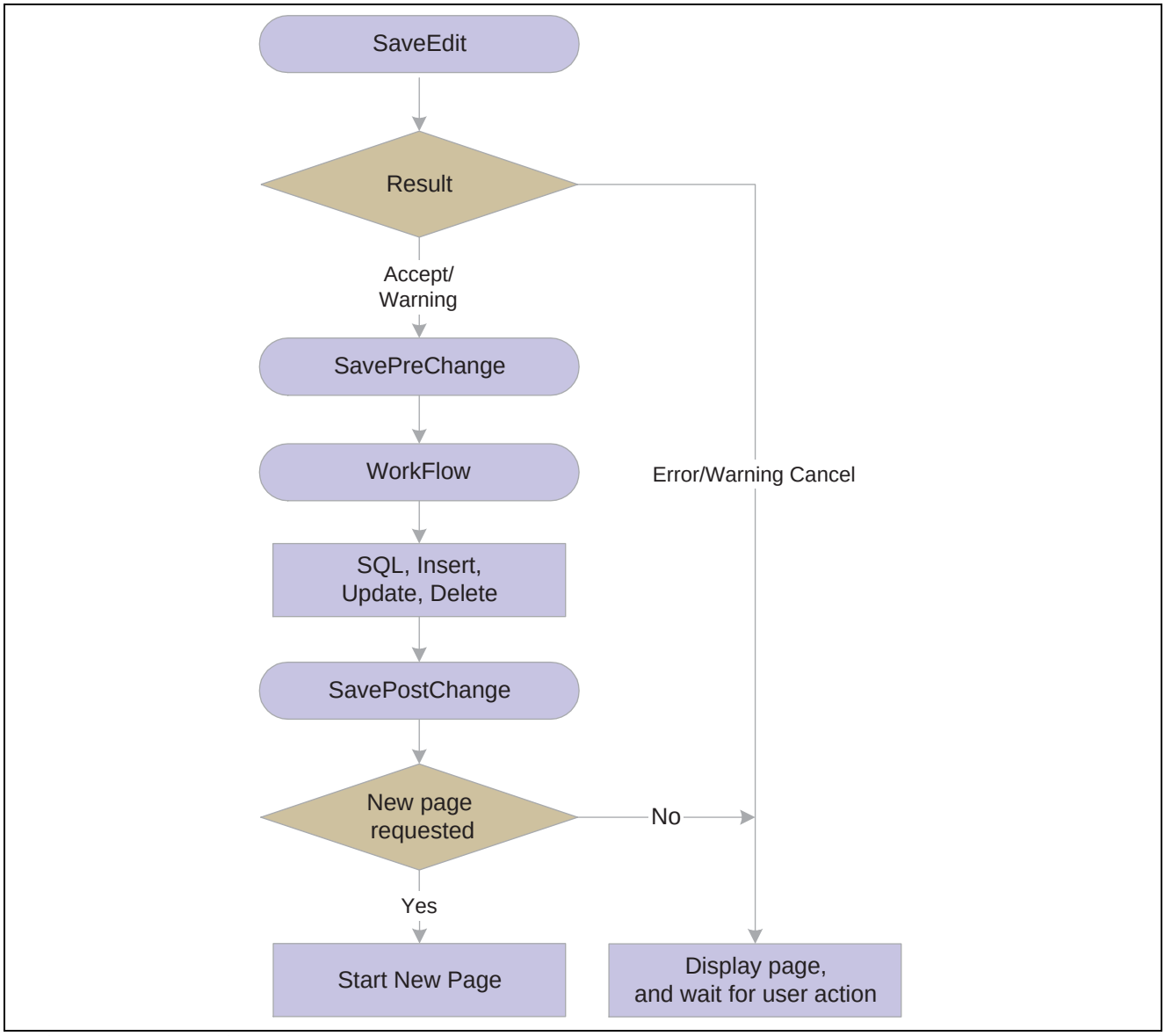
   You can use SavePostChange PeopleCode for processing that must occur after the database update, such as updates to other database tables not in the component buffer.

6. The Component Processor issues a SQL Commit statement to the database server.

7. The Component Processor redisplays the component.

Important! Never use an Error or Warning statement in any save processing event other than SaveEdit. Perform all component data validation in SaveEdit.

The following flow chart shows the logic of this sequence:



Logic of save processing

PeopleSoft Pure Internet Architecture Processing Considerations

Keep the following points in mind concerning the PeopleSoft Pure Internet Architecture:

- If a user changes a field, but there is nothing to cause a trip to the server on that field, default processing and FieldFormula PeopleCode don't run.

  These processes only run when another event causes a trip to the server.

  Other fields that depend on the first field using FieldFormula or default PeopleCode are not updated until the next time there is a server trip.

- In applications that run on the PeopleSoft portal, external, dynamic link information must be placed in RowInit PeopleCode.

  If it's placed in FieldChange PeopleCode, it won't work.

Deferred Processing Mode

When a component runs in deferred processing mode, trips to the server are reduced. When deploying some pages in the browser, you may want the user to be able to input data with minimal interruption or trips to the server. Each trip to the server results in the page being refreshed on the browser, which may cause the display to flicker. It can also slow down your application. By specifying a component as deferred processing mode, you can achieve better performance.

If you've specified deferred processing mode for a component, you can then specify whether a page within a component, or a field on a page, also performs processing in deferred mode. The default is for all pages and components to allow deferred processing. By default, fields do not allow deferred processing.

If you specify that a field or page allows deferred processing, but do not set the component to deferred processing mode, deferred processing mode is not initiated. You must set the component first.

The characteristics of this mode are:

1. Field modification processing is deferred.

   No field modification processing is done on the browser. FieldEdit and FieldChange PeopleCode, as well as other edits, such as required field checks, formats, and so on, do not run until a specific user action occurs. Several actions cause field modification processing to execute, for example, clicking a button or link, navigating to another page in the component, and saving the page. The following actions do not cause field processing:

   - Clicking an external link.
   - Clicking a list (performing a search).
   - Clicking a process button.

     Deferred processing mode affects the appearance of pages in significant ways. For example, related processing is not done when the user presses TAB to move out of a field. Avoid related fields for components that use this mode.

2. Drop-down list box values are static while the page is displayed on the browser.

   Drop-down list box values are generated on the application server when generating the HTML for the page.

   If translate values are used to populate the drop-down list box, and the current record contains an effective date, that date is static while the page is displayed. This means the drop-down list box values may become out of date.

   If prompt table values are used to populate the drop-down list box, the high-order key field values for the prompt table are static while the page is displayed. This means the drop-down list box values may become out of date.

   Avoid interdepencies in drop-down lists used on pages executed in deferred mode, because the lists may quickly become out of date.

3. No field modification processing is done during prompt button processing.

When the user clicks a prompt button, a trip is made to the application server (if values weren't already downloaded) to select the search results from the database and to generate the HTML for the prompt dialog box. During this trip to the application server, field modification processing for the field being prompted is not performed, because this may cause an error message for another field on the page, and this may confuse the user. When there are deferred changes to other fields, field modification processing for these fields is done before prompting. The field modification for the prompted field is done after returning from the prompt page. While the page is displayed, the high-order key field values for the prompt table should be static or not require field modification processing. Display-only drop-down list box, radio button, and check box fields do not require field modification processing. Field values that do not require field modification processing are temporarily written to the component buffer, without any field modification processing being performed on them, including FieldEdit and FieldChange PeopleCode. The system restores the original state of the page processor before returning to the browser.

4. Field modification processing executes in field layout order.

   The entire field modification processing sequence executes in field layout order for each field. If a field passes the system edits and FieldEdit PeopleCode, the field value is written to the component buffer. If an error occurs, field modification processing stops, and the system generates new HTML for the page, with the field in error highlighted and sent to the browser.

5. PeopleCode dependencies between fields on the page do not work as expected.

   Avoid PeopleCode dependencies between fields on pages displayed in deferred processing mode. Also, avoid FieldChange PeopleCode that changes the display.

   The following are examples of PeopleCode dependencies between fields on the page and the application server's action. In the following examples, field A comes before field B, which comes before field C.

   - Field A has FieldChange PeopleCode that hides field B or it makes unavailable for entry.

     The value in field B of the page that was submitted from the browser is discarded.

   - Field B has FieldChange PeopleCode that hides field A or makes it unavailable for entry.

     The change made by the user for field A, if any, remains in the component buffer.

   - Field A has FieldChange PeopleCode that changes the value in the component buffer for field B.

     If the value in field B of the page that was submitted from the browser passes the system edits and FieldEdit PeopleCode, it iswritten to the component buffer, overriding the change made by field A's FieldChange PeopleCode.

   - Field B has FieldChange PeopleCode that changes the value in the component buffer for field A.

     The change made by field B's FieldChange PeopleCode overrides the change made by the user to field A, if any.

   - Field A has FieldChange PeopleCode that unhides field B or makes it available for entry.

     Field B has the value that was already in the component buffer. If the user requests a different page or finishes, the user may not have the opportunity to enter a value into field B, and therefore the value may not be correct.

   - Field B has FieldChange PeopleCode that changes the value in the component buffer for field A, but field C has FieldChange PeopleCode that hides field B or makes it unavailable for entry.

     The change made by field B's FieldChange PeopleCode, a field that is now hidden or unavailable for entry, overrides the change made by the user to field A, if any.

     Avoid such dependencies by moving FieldChange PeopleCode logic from individual fields to save processing for the component or FieldChange PeopleCode on a PeopleCode command button.

6. Not all buttons cause field modification processing to execute.

External links, list (search), and process buttons do not cause field modification processing to execute.

7. You can use a PeopleCode command button to cause field modification processing to execute.

   An application can include a button for the sole purpose of causing field modification processing to execute. The result is a new page showing any display changes that resulted from field modification processing.

   In addition, if the user clicks the Refresh button, or presses ALT + 0, deferred processing is executed.

8. A scroll button (link) causes field modification processing to execute.

PeopleCode Events

This section discusses:

- Activate event.
- FieldChange event.
- FieldDefault event.
- FieldEdit event.
- FieldFormula event.
- ItemSelected event.
- PostBuild event.
- PreBuild event.
- PrePopup event.
- RowDelete event.
- RowInit event.
- RowInsert event.
- RowSelect event.
- SaveEdit event.
- SavePostChange event.
- SavePreChange event.
- SearchInit event.
- SearchSave event.
- Workflow event.

Note. The term *PeopleCode type* is still frequently used, but it does not fit into the PeopleTools object-based, event-driven metaphor. The term *PeopleCode event* should now be used instead. However, it's often convenient to qualify a class of PeopleCode programs triggered by a specific event with the event name; for example, PeopleCode programs associated with the RowInit events are collectively referred to as *RowInit PeopleCode.*

Activate Event

The Activate event is initiated each time that a page is activated, including when a page is first displayed by a user, or if a user presses TAB between different pages in a component. Each page has its own Activate event.

The Activate event segregates PeopleCode that is related to a specific page from the rest of the application's PeopleCode. Place PeopleCode related to page display or page processing, such as enabling a field or hiding a scroll area, in this event. Also, you can use this event for security validation: if an user does not have clearance to view a page in a component, you would put the code for hiding the page in this event.

Note. PeopleSoft builds a page grid one row at a time. Because the Grid class applies to a complete grid, you cannot attach PeopleCode that uses the Grid class to events that occur before the grid is built; the earliest event you can use is the Activate event. The Activate event is not associated with a specific row and record at the point of execution. This means you cannot use functions such as GetRecord, GetRow, and so on, which rely on context, without specifying more context.

Activate PeopleCode can only be associated with pages.

This event is valid only for pages that are defined as standard or secondary. This event is not supported for subpages.

See Also

Chapter 6, "PeopleCode and the Component Processor," Component Build Processing in Update Modes, page 100

Chapter 6, "PeopleCode and the Component Processor," Component Build Processing in Add Modes, page 102

FieldChange Event

Use FieldChange PeopleCode to recalculate page field values, change the appearance of page controls, or perform other processing that results from a field change other than data validation. To validate the contents of the field, use the FieldEdit event.

See Chapter 6, "PeopleCode and the Component Processor," FieldEdit Event, page 117.

The FieldChange event applies to the field and row that just changed.

FieldChange PeopleCode is often paired with RowInit PeopleCode. In these RowInit/FieldChange pairs, the RowInit PeopleCode checks values in the component and initializes the state or value of page controls accordingly. FieldChange PeopleCode then rechecks the values in the component during page execution and resets the state or value of page controls.

To take a simple example, suppose you have a derived/work field called PRODUCT, the value of which is always the product of page field A and page field B. When the component is initialized, you would use RowInit PeopleCode to initialize PRODUCT equal to A × B when the component starts up or when a new row is inserted. You could then attach FieldChange PeopleCode programs to both A and B which also set PRODUCT equal to A × B. Whenever a user changes the value of either A or B, PRODUCT is recalculated.

FieldChange PeopleCode can be associated with record fields and component record fields.

See Also

Chapter 6, "PeopleCode and the Component Processor," Field Modification, page 103

FieldDefault Event

The FieldDefault PeopleCode event enables you to programmatically set fields to default values when they are initially displayed. This event is initiated on all page fields as part of many different processes; however, it triggers PeopleCode programs only when the following conditions are all true:

- The page field is still blank after applying any default value specified in the record field properties.

  This is true if there is no default specified, if a null value is specified, or if a 0 is specified for a numeric field.

- The field has a FieldDefault PeopleCode program.

In practice, FieldDefault PeopleCode normally sets fields by default when new data is being added to the component; that is, in Add mode and when a new row is inserted into a scroll area.

If a field value is changed, whether through PeopleCode or by a user, the IsChanged property for the row is set to True. The exception to this is when a change is done in the FieldDefault or FieldFormula events. If a value is set in FieldDefault or FieldFormula, the row is not marked as changed.

At save time, all newly inserted and changed rows are written to the database. All newly inserted but not changed rows are not written to the database.

You must attach FieldDefault PeopleCode to the field where the default value is being populated.

Note. An error or warning issued from FieldDefault PeopleCode causes a runtime error.

FieldDefault PeopleCode can be associated with record fields and component record fields.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Default Processing, page 92

FieldEdit Event

Use FieldEdit PeopleCode to validate the contents of a field, supplementing standard system edits. If the data does not pass the validation, the PeopleCode program should display a message using the Error statement, which redisplays the page, displaying an error message and turning the field red.

To permit the field edit but alert the user to a possible problem, use a Warning statement instead of an Error statement. A Warning statement displays a warning dialog box with OK and Explain buttons. It permits field contents to be changed and continues processing as usual after the user clicks OK.

If the validation must check for consistency across page fields, then use SaveEdit PeopleCode instead of FieldEdit.

The FieldEdit event applies to the field and row that just changed.

FieldEdit PeopleCode can be associated with record fields and component record fields.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Field Modification, page 103

FieldFormula Event

The FieldFormula event is not currently used. Because FieldFormula PeopleCode initiates in many different contexts and triggers PeopleCode on every field on every row in the component buffer, it can seriously degrade application performance. Use RowInit and FieldChange events rather than FieldFormula.

If a field value is changed, whether through PeopleCode or by a user, the IsChanged property for the row is usually set to True. However, if a value is set in FieldDefault or FieldFormula, the row is not marked as changed.

At save time, all newly inserted and changed rows are written to the database. All newly inserted but not changed rows are not written to the database.

Note. InPeopleSoft Pure Internet Architecture, if a user changes a field, but there is nothing to cause a trip to the server on that field, default processing and FieldFormula PeopleCode do not run. They only run when another event causes a trip to the server.

As a matter of convention, FieldFormula is now often used in FUNCLIB_ (function library) record definitions to store shared functions. However, you can store shared functions in any PeopleCode event.

FieldFormula PeopleCode is only associated with record fields.

ItemSelected Event

The ItemSelected event is initiated whenever a user selects a menu item from a pop-up menu. In pop-up menus, ItemSelected PeopleCode executes in the context of the page field from where the pop-up menu is attached, which means that you can freely reference and change page fields, just as you could from a button.

Note. This event, and all its associated PeopleCode, does not initiate if run from a component interface.

ItemSelected PeopleCode is only associated with pop-up menu items.

See Also

Chapter 6, "PeopleCode and the Component Processor," Selected Item Processing, page 110

PostBuild Event

The PostBuild event is initiated after all the other component build events have been initiated. This event is often used to hide or unhide pages. It's also used to set component variables.

PostBuild PeopleCode is only associated with components.

PreBuild Event

The PreBuild event is initiated before the rest of the component build events. This event is often used to hide or unhide pages. It's also used to set component variables.

Note. If a PreBuild PeopleCode program issues an error or warning, the user is returned to the search page. If the search record has no keys, a blank component page appears.

Also use the PreBuild event to validate data entered in a search page after a prompt list is displayed. For example, after a user selects key values on a search, the PreBuild PeopleCode program runs, catches the error condition, and issues an error message. The user receives and acknowledges the error message. The component is canceled (because of the error), and the user is returned to the search page. PreBuild PeopleCode is only associated with components.

PrePopup Event

The PrePopup event is initiated just before the display of a pop-up menu.

You can use PrePopup PeopleCode to control the appearance of the pop-up menu.

Note. This event, and all its associated PeopleCode, does not initiate if run from a component interface.

PrePopup PeopleCode can be associated with record fields and component record fields.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Pop-Up Menu Display, page 110

RowDelete Event

The RowDelete event is initiated whenever a user attempts to delete a row of data from a page scroll area. Use RowDelete PeopleCode to prevent the deletion of a row (using an Error or Warning statement) or to perform any other processing contingent on row deletion. For example, you could have a page field called Total on scroll area level zero whose value is the sum of all the Extension page fields on scroll area level one. If the user deleted a row on scroll area level one, you could use RowDelete PeopleCode to recalculate the value of the Total field.

The RowDelete event triggers PeopleCode on any field on the row of data that is being flagged as deleted.

Note. RowDelete does not trigger programs on derived/work records.

RowDelete PeopleCode can be associated with record fields and component records.

Deleting All Rows from a Scroll Area

When the last row of a scroll area is deleted, a new, dummy row is automatically added. As part of the RowInsert event, RowInit PeopleCode is run on this dummy row. If a field is changed by RowInit (even if it’s left blank), the row is no longer new, and therefore is not reused by any of the ScrollSelect functions or the Select method. In this case, you may want to move your initialization code from the RowInit event to FieldDefault.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Row Delete Processing, page 108

Chapter 8, “Using Methods and Built-In Functions,” Using Errors and Warnings in RowDelete Events, page 159

RowInit Event

The RowInit event is initiated the first time that the Component Processor encounters a row of data. IUse it to set the initial state of component controls. This occurs during component build processing and row insert processing. It also occurs after a Select or SelectAll Rowset method, or a ScrollSelect or related function, is executed.

Note. Generally, if none of the fields in the new row are changed after the row has been inserted (either by a user pressing ALT+7 or programmatically), when the page is saved, the new row is not inserted into the database. However, if the ChangeOnInit rowset class property is set to False, you can set values for fields a new row in RowInsert or RowInit PeopleCode, and the row won’t be saved.

RowInit is not field-specific: it triggers PeopleCode on all fields and on all rows in the component buffer.

Do not use Error or Warning statements in RowInit PeopleCode: these cause a runtime error.

RowInit PeopleCode is often paired with FieldChange PeopleCode. In these RowInit/FieldChange pairs, the RowInit PeopleCode checks values in the component and initializes the state or value of page controls accordingly. FieldChange PeopleCode then rechecks the values in the component during page execution and resets the state or value of page controls.

To take a simple example, suppose you have a derived/work field called PRODUCT, the value of which is always the product of page field A and page field B. When the component is initialized, use RowInit PeopleCode to initialize PRODUCT equal to A × B when the component starts up or when a new row is inserted. You could then attach FieldChange PeopleCode programs to both A and B, which also sets PRODUCT equal to A × B. Whenever a user changes the value of either A or B, PRODUCT is recalculated.

RowInit PeopleCode can be associated with record fields and component records.

RowInit Exceptions

In certain rare circumstances, the Component Processor does not run RowInit PeopleCode for some record fields. The Component Processor runs RowInit PeopleCode when it loads the record from the database. However, in some cases, the record can be initialized entirely from the keys for the component. When this happens, RowInit PeopleCode is not run.

For RowInit to not run, the following must all be true:

- The record is at level zero.
- Every record field that is present in the data buffers is also present in the keys for the component.

  The Component Processor determines if the field is required by the component. In practice, this usually means that the field is associated with a page field, possibly hidden, for some page of the component. It could also mean that the field is referenced by some PeopleCode program that is attached to an event on some other field of the component.
- Every record field that is present in the data buffers is display-only.

RowInit not running is not considered to be an error. The purpose of RowInit PeopleCode is to complete initialization of data on the row after it has been read from the database. Because the data in this special circumstance is coming from the keylist, it has already been initialized correctly by whatever processing produced the keylist. More general initialization of the component should be done in PostBuild PeopleCode, not RowInit.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Component Build Processing in Add Modes, page 102

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, “Rowset Class,” ChangeOnInit

RowInsert Event

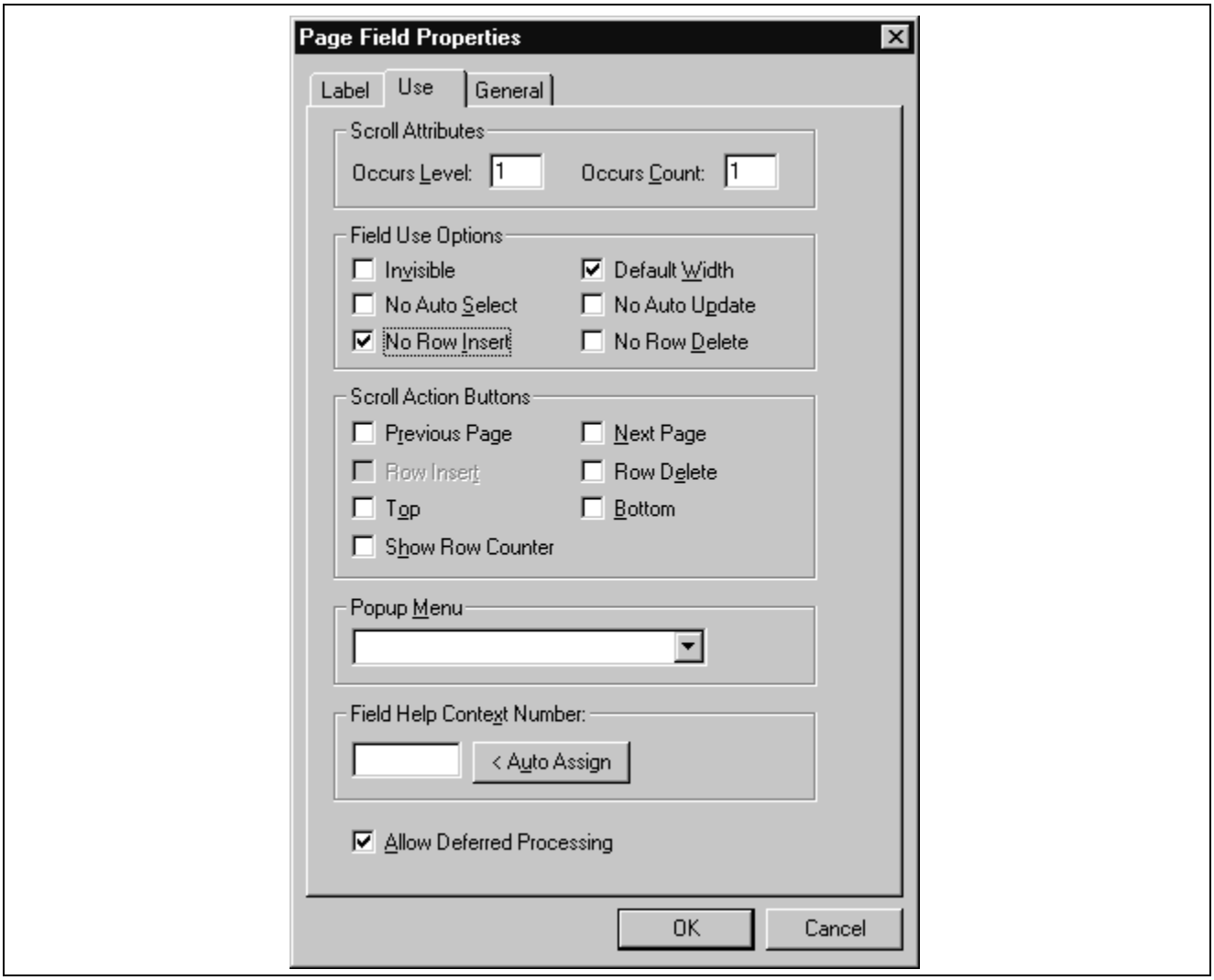
When a user adds a row of data, the Component Processor generates a RowInsert event. You should use RowInsert PeopleCode for processing specific to the insertion of new rows. Do not put PeopleCode in RowInsert that already exists in RowInit, because a RowInit event always initiates after the RowInsert event, which will cause your code to be run twice.

Note. Generally, if none of the fields in the new row are changed after the row has been inserted (either by a user pressing ALT+7 or programmatically), when the page is saved, the new row is not inserted into the database. However, if the ChangeOnInit rowset class property is set to False, you can set values for fields a new row in RowInsert or RowInit PeopleCode, and the row won’t be saved.

The RowInsert event triggers PeopleCode on any field on the inserted row of data.

Do not use a warning or error in RowInsert.

You can prevent a user from inserting rows into a scroll area by selecting the No Row Insert check box in the scroll bar's page field properties, as shown in the following illustration. However, you cannot prevent row insertion conditionally.



Setting row insert properties in page field properties for a scrool bar

Note. RowInsert does not trigger PeopleCode on derived/work fields.

RowInsert PeopleCode can be associated with record fields and component records.

See Also

Chapter 6, "PeopleCode and the Component Processor," Row Insert Processing, page 106

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Rowset Class," ChangeOnInit

RowSelect Event

The RowSelect event is initiated at the beginning of the component build process in any of the update action modes (Update, Update/Display All, Correction). RowSelect PeopleCode is used to filter out rows of data as they are being read into the component buffer. This event also occurs after a ScrollSelect or related function is executed.

A DiscardRow function in RowSelect PeopleCode causes the Component Processor to skip the current row of data and continue to process other rows. A StopFetching statement causes the Component Processor to accept the current row of data, and then stop reading additional rows. If both statements are executed, the program skips the current row of data, and then stops reading additional rows.

PeopleSoft applications rarely use RowSelect, because it's inefficient to filter out rows of data after they've already been selected. Instead, screen out rows of data using search record views and effective-dated tables, which filter out the rows before they're selected. You could also use a ScrollSelect or related function to programmatically select rows of data into the component buffer.

In previous versions of PeopleTools, the Warning and Error statements were used instead of DiscardRow and StopFetching. Warning and Error statements still work as before in RowSelect, but their use is discouraged.

Note. In RowSelect PeopleCode, you can refer to record fields only on the record that is currently being processed. This event, and all its associated PeopleCode, does not initiate if run from a component interface.

RowSelect PeopleCode can be associated with record fields and component records.

See Also

Chapter 6, "PeopleCode and the Component Processor," Row Select Processing, page 101

SaveEdit Event

The SaveEdit event is intiated whenever a user attempts to save the component. You can use SaveEdit PeopleCode to validate the consistency of data in component fields. Whenever a validation involves more than one component field, you should use SaveEdit PeopleCode. If a validation involves only one page field, use FieldEdit PeopleCode.

SaveEdit is not field-specific: it triggers associated PeopleCode on every row of data in the component buffers, except rows flagged as deleted.

An Error statement in SaveEdit PeopleCode displays a message and redisplays the component without saving data. A Warning statement enables the user to click OK and save the data, or to click Cancel and return to the component without saving.

Use the SetCursorPos function to set the cursor position to a specific page field following a warning or error in SaveEdit, to show the user the field (or at least one of the fields) that is causing the problem. Make sure to call SetCursorPos before the error or warning, because these may terminate the PeopleCode program.

SaveEdit PeopleCode can be associated with record fields and components.

See Also

Chapter 6, "PeopleCode and the Component Processor," Save Processing, page 110

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

SavePostChange Event

After the Component Processor updates the database, it initiates the SavePostChange event. You can use SavePostChange PeopleCode to update tables not in your component using the SQLExec built-in function.

An error or warning in SavePostChange PeopleCode causes a runtime error. Avoid errors and warnings in this event.

The system issues a SQL Commit statement after SavePostChange PeopleCode completes successfully.

If you are executing Workflow PeopleCode, keep in mind that if the Workflow PeopleCode fails, SavePostChange PeopleCode is not executed. If your component has both Workflow and SavePostChange PeopleCode, consider moving the SavePostChange PeopleCode to SavePreChange or Workflow.

If you are doing messaging, your Publish PeopleCode should go into this event.

SavePostChange does not execute if there is an error during the save. For example, if there is a data conflict error because another user updated the same data at the same time, SavePostChange does not execute.

Important! Never issue a SQL Commit or Rollback statement manually from within a SQLExec function. Let the Component Processor issue these SQL commands.

SavePostChange PeopleCode can be associated with record fields, components, and component records.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Save Processing, page 110

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” SQLExec

SavePreChange Event

The SavePreChange event is initiated after SaveEdit completes without errors. SavePreChange PeopleCode provides one final opportunity to manipulate data before the system updates the database; for instance, you could use SavePreChange PeopleCode to set sequential high-level keys. If SavePreChange runs successfully, a Workflow event is generated, and then the Component Processor issues appropriate Insert, Update, or Delete SQL statements.

SavePreChange PeopleCode is not field-specific: it triggers PeopleCode on all fields and on all rows of data in the component buffer.

SavePreChange PeopleCode can be associated with record fields, components, and component records.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Save Processing, page 110

SearchInit Event

The SearchInit event is generated just before a search, add, or data-entry dialog box is displayed. SearchInit triggers associated PeopleCode in the search key fields of the search record. This enables you to control processing before a user enters values for search keys in the dialog box. In some cases, you may want to set the value of the search dialog fields programmatically. For example, the following program in SearchInit PeopleCode on the component search key record field EMPLID sets the search key page field to the user’s employee ID, makes the page field unavailable for entry, and enables the user to modify the user’s own data in the component:

```
EMPLID = %EmployeeId;
Gray (EMPLID);
AllowEmplIdChg(true);
```

You can activate system defaults and system edits in the search page by calling SetSeachDefault and SetSearchEdit in SearchInit PeopleCode. You can also control the behavior of the search page, either forcing it to appear even if all the required keys have been provided, or by skipping it if possible, with the SetSeachDialogBehavior function. You can also force search processing to always occur by selecting the Force Search Processing check box in the component properties in PeopleSoft Application Designer.

Note. This event, and all its associated PeopleCode, do not inititiate if run from a component interface.

SearchInit PeopleCode can be associated with record fields and component search records.

SearchInit PeopleCode Function Restrictions

You can't use the following functions in SearchInit PeopleCode:

- DoModal
- DoModalComponent
- Transfer
- TransferPage

See Also

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

Chapter 6, "PeopleCode and the Component Processor," Search Processing in Update Modes, page 95

Chapter 6, "PeopleCode and the Component Processor," Search Processing in Add Modes, page 98

SearchSave Event

SearchSave PeopleCode is executed for all search key fields on a search, add, or data-entry dialog box after a user clicks Search. This enables you to control processing after search key values are entered, but before the search based on these keys is executed. A typical use of this feature is to provide cross-field edits for selecting a minimum set of key information. This event is also used to force a user to enter a value in at least one field, even if it's a partial value, to help narrow a search for tables with many rows.

Note. SearchSave is not initiated when values are selected from the search list. To validate data entered in the search page, use the Component PreBuild event.

You can use Error and Warning statements in SearchSave PeopleCode to send the user back to the search page if the user entry does not pass validations implemented in the PeopleCode.

Note. This event, and all its associated PeopleCode, is not initiated if run from a component interface.

SearchSave PeopleCode can be associated with record fields and component search records.

Note. Do not use the %Menu system variable in this event. You may get unexpected results.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Search Processing in Update Modes, page 95

Chapter 6, “PeopleCode and the Component Processor,” Search Processing in Add Modes, page 98

Workflow Event

Workflow PeopleCode executes immediately after the SavePreChange event and before the database update that precedes the SavePostChange event. The Workflow event segregates PeopleCode related to workflow from the rest of the application’s PeopleCode. Only PeopleCode related to workflow (such as TriggerBusinessEvent) should be in workflow programs. Your program should deal with the Workflow event only after any SavePreChange processing is complete.

Workflow PeopleCode is not field-specific: it triggers PeopleCode on all fields and on all rows of data in the component buffer.

WorkFlow PeopleCode can be associated with record fields and components.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Save Processing, page 110

Enterprise PeopleTools 8.45 PeopleBook: Workflow Technology, “Defining Event Triggers,” Writing Workflow PeopleCode

PeopleCode Execution in Pages with Multiple Scroll Areas

Components with multiple levels can have multiple rows of data from multiple primary record definitions. You must know the order in which the system processes buffers for this data, because it applies PeopleCode in the same order.

The Component Processor uses a depth-first algorithm to process rows in multiple-scroll-area pages, starting with a row at level zero, drilling down to dependent rows on lower levels, and then working up the hierarchy until the system has processed all the dependent rows of the last row on the highest level.

Scroll Level One

When pages have only one scroll bar, the Component Processor processes processes record definitions at scroll level zero, then all rows of data at scroll level one.

Data is retrieved for all rows with a single Select statement, and then merged with buffer structures.

Scroll Level Two

With scroll bars at multiple scroll levels, the system processes a single row of data at scroll level one, then processes all its subordinate rows of data at scroll level two. After processing all subordinate data at scroll level two, it processes the next row for scroll level one, and all the subordinate data for that row. The system continues in this fashion until it processes everything.

Scroll Level Three

The Component Processor uses the same method for processing subordinate data at scroll level three. Data is retrieved for all rows with a single Select statement, and then merged with buffer structures. the Component Processor processes a single row of data at scroll level two, it processes all subordinate data before processing the next level-two row.

See Also

Chapter 4, “Referencing Data in the Component Buffer,” Understanding Component Buffer Structure and Contents, page 41

CHAPTER 7

PeopleCode and PeopleSoft Pure Internet Architecture

The chapter discusses how to:

- Considerations using PeopleCode in PeopleSoft Pure Internet Architecture.
- Using PeopleCode with PeopleSoft Pure Internet Architecture
- Call dynamic link library (DLL) functions on the application server.
- Update the Installation and PSOPTIONS tables.

Considerations Using PeopleCode in PeopleSoft Pure Internet Architecture

Take the following considerations into account when writing PeopleCode programs for PeopleSoft Pure Internet Architecture:

- To help your application run efficiently, avoid using field-level PeopleCode events (FieldEdit and FieldChange).

  Each field-level PeopleCode program requires a trip to the application server.

  The majority of PeopleCode programs run on the application server as part of the component build and save process. Do not hesitate to use PeopleCode for building and saving components.

- If a user changes a field, but there is nothing to cause a trip to the server on that field, default processing and FieldFormula PeopleCode do not run.

  This processing occurs only when another event causes a trip to the server.

  Other fields that depend on the first field using FieldFormula or default PeopleCode are not updated until the next time there is a server trip.

- In applications that run on the PeopleSoft portal, external dynamic link information must be placed in RowInit PeopleCode.

  If external dynamic link information is placed in FieldChange PeopleCode, it won't work.

- Trips to the server are reduced when a component runs in deferred processing mode.

  Each trip to the server results in the page being completely refreshed on the browser, which may cause the display to flicker. It can also slow down your application. Deferred processing mode results in better performance.

See Also

Chapter 6, “PeopleCode and the Component Processor,” Deferred Processing Mode, page 113

Using PeopleCode with PeopleSoft Pure Internet Architecture

As a licensee of PeopleTools, you are licensed to use the base portal technology, which is limited to navigation to licensed PeopleSoft applications. If you want to register additional non-PeopleSoft content, customize your homepage, or create any pagelets, you must license PeopleSoft Enterprise Portal.

This section discusses how to:

- Use internet scripts.
- Use the field object Style property.
- Use the HTML area.
- Use HTML definitions and the GetHTML Text function.
- Use PeopleCode to Populate Key Fields in Search Dialog Boxes

Using Internet Scripts

An internet script is a specialized PeopleCode function that generates dynamic web content. Internet Scripts interact with web clients (browsers) using a request-response paradigm based on HTTP.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, “Internet Script Classes (iScript)”

Using the Field Object Style Property

In PeopleSoft Application Designer, on the Use tab of the page definition properties, you can associate a page with a style sheet component.

The style sheet has several classes of styles defined for it. You can edit each style class to change the font, the color, the background, and so on. Then you can dynamically change the style of a field using the Style field class property. This does not change the style sheet, only the style class associated with that field.

The following example changes the style class of a field depending on a value entered by the user. This code is in the FieldChange event.

```
Local Field &field;

&field = GetField();

If TESTFIELD1 = 1 Then;
   &field.Style = "PSHYPERLINK";
End-If;

If TESTFIELD1 = 2 Then;
   &field.Style = "PSIMAGE";
End-If;
```

The following illustrations show the fields with different styles:

Field with PSHYPERLINK style

Field with PSIMAGE style

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer, "Creating Style Sheet Definitions"

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

Using the HTML Area

There are two methods to populate an HTML area control. Both require accessing the HTML area in the PeopleSoft Application Designer. One method is to select Constant on the HTML tab of the HTML page field properties dialog and enter HTML directly into the page field dialog.

The other method is to select Value on the HTML tab of the HTML page field properties dialog and associate the control with a record field. At runtim, populate that field with the text you want displayed in the HTML area.

If you are using an HTML area to add form controls to a page, you can use GetParameter request class method in PeopleCode to get the user input from those controls.

Note. When you associate an HTML area control with a field, make sure the field is long enough to contain the data you want to pass to it. For example, if you associate an HTML area control with a field that is only 10 characters long, only the first 10 characters of your text is displayed.

The following code populates an HTML area with a simple bulleted list. This code is in the RowInit event of the record field associated with the HTML control.

```
Local Field &HTMLField;

&HTMLField = GetField();
&HTMLField.Value = "<ul><li>Item one</li><li>Item two</li></ul>";
```

The following code is in the FieldChange event of a button. It populates an HTML area (associated with the record field CHART_DATA.HTMLAREA) with a simple list.

```
Local Field &HTMLField;

&HTMLField = GetRecord(Record.CHART_DATA).HTMLAREA;
&HTMLField.Value = "<ul><li>Item one</li><li>Item two</li></ul>";
```

The following code populates an HTML area (associated with the record DERIVED_HTML and the field HTMLAREA) with the output of the GenerateTree function:

```
DERIVED_HTML.HTMLAREA = GenerateTree(&TREECTL);
```

The following tags are unsupported by the HTML area control:

- Body
- Frame
- Frameset
- Form
- Head
- HTML
- Meta
- Title

See Also

Chapter 9, “Understanding HTML Trees and the GenerateHTML Function,” Using the GenerateTree Function, page 165

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer, “Creating HTML Definitions”

Using HTML Definitions and the GetHTMLText Function

If you’re using the same HTML text in more than one place, or if it’s a large, unwieldy string, you can create an HTML definition in PeopleSoft Application Designer, then use the GetHTMLText function to populate an HTML area control.

The following is the HTML string to create a simple table:

```
<P>
<TABLE>

  <TR bgColor=#008000>
    <TD>
      <P><FONT color=#f5f5dc face="Arial, Helvetica, sans-serif"
      size=2>message 1 </FONT></P></TD></TR>
  <TR bgColor=#0000cd>
    <TD>
      <P><FONT color=#00ffff face="Arial, Helvetica, sans-serif"
      size=2>message 2</FONT></P></TD></TR>
   </TABLE></P>
```

This HTML is saved to an HTML definition called TABLE_HTML.

This code is in the RowInit event of the record field associated with the HTML area control:

```
Local Field &HTMLField;

&HTMLField = GetField();
&string = GetHTMLText(HTML.TABLE_HTML);
&HTMLField.Value = &string;
```

This code produces the following:

HTML definition example

See Also

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

Using HTML Definitions and the GetJavaScriptURL Method

HTML definitions can contain JavaScript programs in addition to HTML. If you have an HTML definition that contains JavaScript, use the GetJavaScriptURL Response method to access and execute the script.

This example assumes the existence in the database of a HTML definition called HelloWorld_JS that contains some JavaScript:

```
Function IScript_TestJavaScript()

   %Response.WriteLine("<script src= " |
%Response.GetJavaScriptURL(HTML.HelloWorld_JS) | "></script>");

End-Function;
```

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Internet Script Classes (iScript)," GetJavaScriptURL

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer, "Creating HTML Definitions"

Using PeopleCode to Populate Key Fields in Search Dialog Boxes

In a PeopleSoft Pure Internet Architecture application, you typically want users to directly access their own data. To facilitate this, you may want to use SearchInit PeopleCode to populate standard key fields in search page fields and then make the fields unavailable for entry. You might assign the search key field a default value based on the user ID or alias the user entered when signing in.

You must also call the AllowEmplIdChg function, which enables users to change their own data. This function takes a single Boolean parameter in which you pass True to allow employees to change their own data.

Here is a simple example of such a SearchInit program, using %EmployeeId to identify the user:

```
EMPLID = %EmployeeId;
Gray (EMPLID);
AllowEmplIdChg(true);
```

Calling DLL Functions on the Application Server

To support processes running on an application server, you can declare and call functions compiled in Microsoft Windows dynamic link libraries and in UNIX shared libraries (or shared objects, depending on the specific UNIX platform). You can do this either with a special PeopleCode declaration, or using the business interlink framework.

When you call out to a DLL using PeopleCode, on Microsoft Windows NT application servers, the DLL file has to be on the path. On UNIX application servers, the shared library file must be on the library path (as defined for the specific UNIX platform).

The PeopleCode declaration and function call syntax remains unchanged. For example, the following PeopleCode could be used to declare and call a function LogMsg in an external library Testdll.dll on a Microsoft Windows client or a Windows application server, or a libtestdll.so on an UNIX application server. The UNIX shared library's extension varies by the specific UNIX platform.

```
Declare Function LogMsg Library "testdll" (string, string)
       Returns integer;

&res = LogMsg("\temp\test.log", "This is a test");
```

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Business Interlinks, "Getting Started with PeopleSoft Business Interlinks"

Sample Cross-Platform External Test Function

Following is the C source code for a sample cross-platform test file. It is a basic function that opens a log file and appends a line to it. If you compile the code using a C++ compiler, the functions must be declared using external C, to ensure C-language linkage.

This file contains an interface function required for non-Microsoft-Windows environments. This function is compiled only when compiling for a non-Windows environment (for example, UNIX). The interface function references a provided header file, pcmext.h. The interface function is passed type codes that can be optionally used for parameter checking.

```
/*
 * Simple test function for calling from PeopleCode.
 * This is passed two strings, a file name and a message.
 * It creates the specified file and writes the message
 * to it.
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#ifdef _WINDOWS
#define DLLEXPORT __declspec(dllexport)
#define LINKAGE __stdcall
#else
#define DLLEXPORT
#define LINKAGE
```

```
#endif

DLLEXPORT int LINKAGE LogMsg(char * fname, char * msg);

/*****************************************************
*  PeopleCode External call test function.           *
*                                *
*  Parameters are two strings (filename and message)    *
*  Result is 0 if error, 1 if OK                *
*                                *
*                                *
*  To call this function, the following PeopleCode is   *
*  used                              *
*                               *
*  Declare Function LogMsg Library "testdll"         *
*  (string, string)                        *
*       Returns integer;                    *
*                                      *
*  &res = LogMsg("\temp\test.log", "This is a test");   *
*                               *
*****************************************************/

DLLEXPORT int LINKAGE LogMsg(char * fname, char * msg)
{
    FILE *fp;

    fp = fopen(fname, "a");         /* append */
    if (fp == NULL) return 0;

    fprintf(fp, "%s\n", msg);
    fclose(fp);
    return 1;
}


#ifndef _WINDOWS

/*****************************************************
*  Interface function.                          *
*                               *
*  This is not needed for Windows....                   *
*                               *
*****************************************************/

#include  "pcmext.h"
#include  "assert.h"

void LogMsg_intf(int nParam, void ** ppParams, EXTPARAMDESC * pDesc)
{
    int    rc;
```

```
    /* Some error checking */
    assert(nParam == 2);
    assert(pDesc[0].eExtType == EXTTYPE_STRING
       && pDesc[1].eExtType == EXTTYPE_STRING
       && pDesc[2].eExtType == EXTTYPE_INT);

    rc = LogMsg((char *)ppParams[0],
         (char *)ppParams[1]);
    *(int *)ppParams[2] = rc;

}

#endif
```

Updating the Installation and PSOPTIONS Tables

When an application updates either the PSOPTIONS or the Installation table it must call UpdateSysVersion from the SavePreChange PeopleCode event. This way, updates take effect at the next page load. Otherwise, the change does not take effect at the client workstation until the user signs out and signs back in.

Important! Only a database administrator or the equivalent should change these tables.

CHAPTER 8

Using Methods and Built-In Functions

This chapter provides an overview of restrictions on method and function use and discusses how to:

- Implement modal transfers.
- Implement the multi-row insert feature.
- Use the ImageReference field.
- Insert rows using PeopleCode.
- Use object linking and embedding (OLE) functions.
- Use the Select and SelectNew methods.
- Use standalone rowsets.
- Use errors and warnings.
- Use the RemoteCall feature.

See Also

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

Understanding Restrictions on Method and Function Use

This section discusses:

- Think-time functions.
- WinMessage and MessageBox functions.
- Program execution with fields not in the data buffer.
- Errors and warnings.
- DoSave function.
- Record class database methods.
- SQL class methods and functions.
- Component interface restricted functions.
- SearchInit PeopleCode function restrictions.
- CallAppEngine function.
- ReturnToServer function.
- GetPage function.

- GetGrid function.
- Publish method.
- SyncRequest method.

Think-Time Functions

Think-time functions suspend processing either until the user has taken some action (such as clicking a button in a message box) or until an external process has run to completion (for example, a remote process).

Avoid think-time functions in the following PeopleCode events:

- SavePreChange.
- Workflow.
- RowSelect.
- SavePostChange.
- Any PeopleCode event that executes as a result of a ScrollSelect, ScrollSelectNew, RowScrollSelect, or RowScrollSelectNew function call.
- Any PeopleCode event that executes as a result of a Select or SelectNew rowset method.

Violation of this rule can result in application failure.

The following are think-time functions:

- Calls to an external DLL.
- DoCancel.
- DoModal.
- DoModalComponent.
- Exec (this is think-time only when synchronous).
- File attachment functions.
- InsertImage.
- Object functions, such as CreateObject, ObjectDoMethod, ObjectSetProperty, and ObjectGetProperty (these are think-time only when the object requires user action).
- Prompt.
- RemoteCall.
- RevalidatePassword.
- WinExec (think-time only when synchronous).
- WinMessage and MessageBox (depending on the *style* parameter).

WinMessage and MessageBox Functions

The WinMessage and MessageBox functions sometimes behave as think-time functions, depending on the value passed in the function's *style* parameter, which controls, among other things, the number of buttons displayed in the message dialog box.

Note. The *style* parameter is ignored if the message has any severity other than Message.

Here is the syntax of both functions:

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

Note. The WinMessage function is supported for compatibility with previous releases of PeopleTools. New applications should use MessageBox instead.

If the *style* parameter specifies more than one button, the function behaves as a think-time function and is subject to the same restrictions as other think-time functions (that is, it should never be used from SavePreChange through SavePostChange PeopleCode, or in RowSelect).

If the *style* parameter specifies a single button (that is, the OK button), then the function can be called in any PeopleCode event.

Note. In the Microsoft Windows client, MessageBox dialog boxes include an Explain button to display more detailed information stored in the message catalog. The presence of the Explain button has no bearing on whether a message box behaves as a think-time function.

The *style* parameter is optional in WinMessage. If *style* is omitted, WinMessage displays OK and Cancel buttons, which causes the function to behave as a think-time function. To avoid this situation, always pass an appropriate value in the WinMessage *style* parameter.

The following table shows the values that can be passed in the *style* parameter. To calculate the value to pass, make one selection from each category in the table, then add the selections.

Category	Value	Constant	Meaning
Buttons	0	%MsgStyle_OK	The message box contains one button: OK.
Buttons	1	%MsgStyle_OKCancel	The message box contains two buttons: OK and Cancel.
Buttons	2	%MsgStyle_ AbortRetryIgnore	The message box contains three buttons: Abort, Retry, and Ignore.
Buttons	3	%MsgStyle_ YesNoCancel	The message box contains three buttons: Yes, No, and Cancel.
Buttons	4	%MsgStyle_YesNo	The message box contains two buttons: Yes and No.
Buttons	5	%MsgStyle_RetryCancel	The message box contains two buttons: Retry and Cancel.

Note. The following values for *style* can only be used in the Microsoft Windows client. They have no affect in PeopleSoft Pure Internet Architecture.

Category	Value	Constant	Meaning
Default Button	0	%MsgDefault_First	The first button is the default.
Default Button	256	%MsgDefault_Second	The second button is the default.
Default Button	512	%MsgDefault_Third	The third button is the default.
Icon	0	%MsgIcon_None	None
Icon	16	%MsgIcon_Error	A stop-sign icon appears in the message box.
Icon	32	%MsgIcon_Query	A question-mark icon appears in the message box.
Icon	48	%MsgIcon_Warning	An exclamation-point icon appears in the message box.
Icon	64	%MsgIcon_Info	An icon consisting of a lowercase letter *i* in a circle appears in the message box.

See Also

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

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

Program Execution with Fields Not in the Data Buffer

Under certain conditions, when you access a field that is not in the data buffer, a portion of your PeopleCode program is skipped. The skip occurs when:

- The reference is in the Import Manager.
- The reference is from the FieldDefault or FieldFormula events.

After the call to the invalid field, execution skips to the next top-level statement. Top-level statements are are not nested inside other statements. The start of a PeopleCode program is a top-level statement. Nesting begins with the first conditional statement (such as While or If) or the first function call.

For example, if your code is executing in a function and inside an If…then…end-if statement, and it runs into the skip conditions, the next statement executed is the one after the End-if statement, still inside the function.

Errors and Warnings

Errors and warnings should not be used in FieldDefault, FieldFormula, RowInit, FieldChange, RowInsert, SavePreChange, WorkFlow, and SavePostChange PeopleCode events. An error or warning in these events causes a runtime error that forces cancellation of the component.

See Also

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

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

DoSave Function

Use DoSave only in FieldEdit, FieldChange, or MenuItemSelected PeopleCode events.

See Also

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

Record Class Database Methods

You use the following record class methods to update the database:

- Delete
- Insert
- Save
- Update

Only use these methods in the following events (events that allow database updates):

- SavePreChange
- WorkFlow
- SavePostChange
- Message Subscription
- FieldChange
- Application Engine PeopleCode action

See Also

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

SQL Class Methods and Functions

Use the SQL class to update the database. Use these functions and methods only in the following events (events that allow database updates):

- SavePreChange
- WorkFlow
- SavePostChange
- Message Subscription
- FieldChange
- Application Engine PeopleCode action

Component Interface Restricted Functions

PeopleCode events and functions that relate exclusively to graphical user interfaces and online processing cannot be used by component interfaces. These include:

- Menu PeopleCode and pop-up menus.

  The ItemSelected and PrePopup PeopleCode events are not supported. In addition, the DisableMenuItem, EnableMenuItem, and HideMenuItem functions are not supported.

- Transfers between components, including modal transfers.

  The DoModal, EndModal, IsModal, Transfer, TransferPage, DoModalComponent, and IsModalComponent functions cannot be used.

- Cursor positioning by the SetCursorPos function.
- The SetControlValue function.
- The WinMessage function.

When executed using a component interface, these functions do nothing and return a default value.

Using the Transfer function terminates the current PeopleCode program.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Component Interfaces, "Programming Component Interfaces in PeopleCode"

SearchInit PeopleCode Function Restrictions

You can't use the following functions in SearchInit PeopleCode:

- DoModal
- DoModalComponent
- Transfer
- TransferPage

CallAppEngine Function

Use the CallAppEngine function only in events that allow database updates, because, generally, if you're calling PeopleSoft Application Engine, you're intending to perform database updates. This includes the following PeopleCode events:

- SavePreChange (Page)
- SavePostChange (Page)
- Workflow
- Message Subscription
- FieldChange

CallAppEngine cannot be used in a PeopleSoft Application Engine PeopleCode action. If you need to access one Application Engine program from another Application Engine program, use the CallSection action.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Engine, "Creating Application Engine Programs"

ReturnToServer Function

The ReturnToServer function returns a value from a PeopleCode application messaging program to the publication or subscription server. You would use this in either your publication or subscription routing code, not in one of the standard Component Processor events.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker, "Defining Message Channels and Messages"

GetPage Function

The GetPage function cannot be used until after the Component Processor has loaded the page. You should not use this function in an event prior to the PostBuild event.

See Also

Chapter 6, "PeopleCode and the Component Processor," page 81

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

GetGrid Function

PeopleSoft builds a grid one row at a time. Because the Grid class applies to a complete grid, you cannot use the GetGrid function in an event prior to the Activate event.

See Also

Chapter 6, "PeopleCode and the Component Processor," page 81

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

Publish Method

If you are using PeopleSoft Integration Broker, your sending PeopleCode should go in the SavePostChange event, for either the record or the component.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker, "Defining Message Channels and Messages"

SyncRequest Method

If you are using PeopleSoft Integration Broker, your SyncRequest PeopleCode should go in the SavePostChange event, for either the record or the component.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker, "Defining Message Channels and Messages"

Implementing Modal Transfers

This section provides an overview of modal transfers and discusses how to implement modal transfers.

Understanding Modal Transfers

When you use modal transfers to transfer from one component (the *originating* component) to another component (the *modal* component), the user must click the OK or Cancel buttons on the modal component before returning to the originating component.

Modal transfers provide some control over the order in which the user fills in pages, which is useful where data in the originating component can be derived from data entered by the user into the modal component.

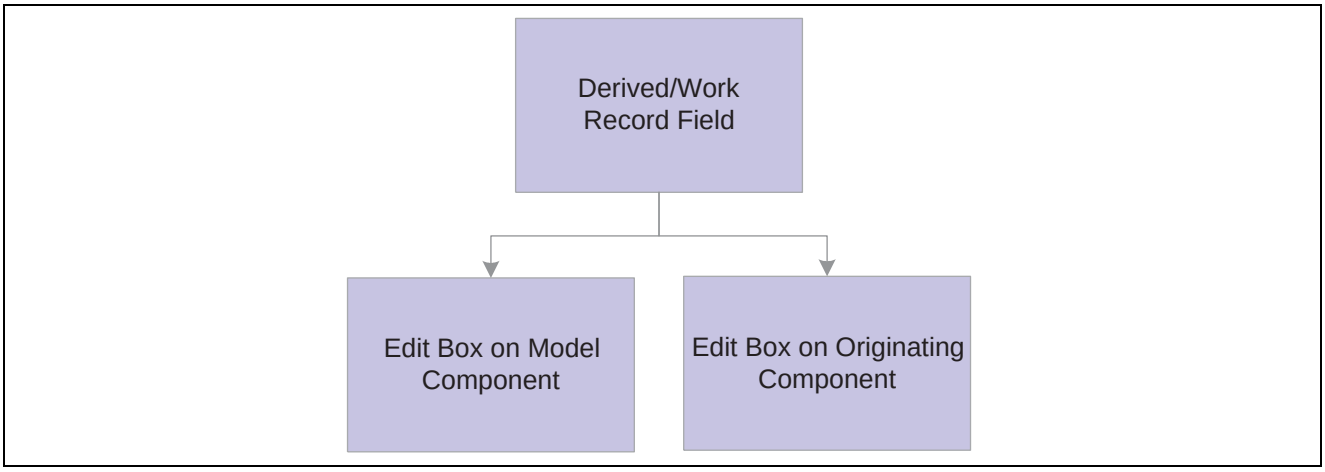
Limit use of this feature, as it forces users to complete interaction with the modal page before returning to the main component.

Note. Modal transfers cannot be initiated from SearchInit PeopleCode.

A modal component resembles a Microsoft Windows modal dialog box. It displays three buttons: OK, Cancel, and Apply. No toolbars or windows are available while the modal component has the focus. The OK button saves changes to the modal component and returns the user to the originating component. The Apply button saves changes to the modal component without returning to the originating component. The Cancel button returns the user to the originating component without saving changes to the modal component.

Modal components are generally smaller than the page from which they are invoked. Remember that OK and Cancel buttons are added at runtime, thus increasing the size of the pages.

The originating component and the modal component share record fields in a derived/work record called a *shared work record.* The derived/work fields of this record provide the two components with an area in memory where they can share data. Edit boxes in both components are associated with the same derived/work field, so that changes made to this field in the originating component are reflected in the modal component, and vice versa. The following diagram illustrates this shared memory:



Edit boxes on the originating and modal components share the same data

Edit boxes associated with the same derived/work fields must be placed at level zero in both the originating component and the modal component.

You can use the shared fields to:

- Pass values assigned to the search keys in the modal component search record.

  If these fields are missing or invalid, the search page appears, enabling the user to enter search keys.
- Pass other values from the originating component to the modal component.
- Pass values back from the modal component to the originating component.

Implementing Modal Transfers

Any component accessible through an application menu system can be accessed using a modal transfer. However, to implement a modal transfer, you must modify pages in both the originating component and the modal component. After these modifications are complete, you can implement the modal transfer using the DoModalComponent function from a page in the originating component.

Before beginning this process, you should answer the following questions:

- Should the originating component provide search key values for the modal component?

  If so, what are the search keys? (Check the modal component's search record.)
- Does the originating component need to pass any data to the modal component?

  If so, what record fields are needed to store this data?
- Does the modal component need to pass any data back to the originating component?

  If so, what record fields are needed to store this data?

To implement a modal transfer:

1. Create derived/work record fields for sharing data between the originating and modal components.

   Create a new derived/work record or open an existing derived/work record. If suitable record fields exist, you can use them; otherwise create new record fields for any data that needs to be shared between the components. These can be search keys for the modal component, data to pass to the modal component, or data to pass back to the originating component.
2. Add derived work fields to the level-zero area of the originating component.

Add one edit box for each of the derived/work fields that you need to share between the originating and modal components to the level-zero area of the page from which the transfer will take place. You probably want to make the edit boxes invisible.

3. Add the same derived work fields to the level-zero area of the modal component.

   Add one edit box for each of the edit boxes that you added in the previous step to the level-zero area of the page to which you are transferring. You probably want to make the edit boxes invisible.

4. Add PeopleCode to pass values into the derived/work fields in the originating component.

   To provide search key values or pass data to the modal page, write PeopleCode that assigns appropriate values to the derived/work fields before DoModalComponent is called.

   For example, if the modal component search key is PERSONAL_DATA.EMPLID, you could place the following assignment statement in the derived/work field's RowInit event:

   ```
   EMPLID = PERSONAL_DATA.EMPLID
   ```

   You also might assign these values in the same program where DoModalComponent is called.

5. Add PeopleCode to access and change the derived/work fields in the modal component.

   No PeopleCode is required to pass search key values during the search. However, if other data has been passed to the modal component, you may need PeopleCode to access and use the data. You may also need to assign new values to the shared fields so that they can be used by the originating component.

   It is possible that the component was accessed through the menu system and not through a modal transfer. To write PeopleCode that runs only in the component when it is running modally, use the IsModalComponent function:

   ```
   If IsModalComponent() Then
      /* PeopleCode for modal execution only.  */
   End-If
   ```

6. Add PeopleCode to access changed derived/work fields in the originating component.

   If the modal component has altered the data in the shared work fields, you can write PeopleCode to access and use the data after DoModalComponent has executed.

Note. You can use the EndModalComponent function as a programmatic implementation of the Ok and Cancel buttons.

See Also

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

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

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

Implementing the Multi-Row Insert Feature

Enabling the multi-row insert feature in grids or scroll areas can reduce response times for transactions that usually require entering many rows of data. With the multi-row feature, users specify the number of rows to add to a grid or scroll area, and empty rows appear for data entry.

This feature cannot be used with effective-dated grids or scroll areas. In addition, the feature may not apply if the entire row is populated using PeopleCode, especially if the data is copied from prior rows. If the feature does apply in this case, the default value of the ChangeOnInit property can be used (the default value is True, which means any PeopleCode updates done in the RowInit or RowInsert events set the IsChanged and IsNew properties to True).

To use the multi-row insert feature:

1. Specify deferred mode processing.

   The multi-row feature reduces transaction times by eliminating excess server trips. To take full advantage of this feature, the transaction should be set to execute in deferred mode. Deferred mode should be set for the component, all pages in the component, and all fields on those pages.

2. Enable the multi-row feature.

   For each grid or scroll area where appropriate, select the Allow Multi-row Insert check box under the Use tab in the grid or scroll area property sheet.

3. Add ChangeOnInit PeopleCode.

   Setting the ChangeOnInit property for a rowset to False enables PeopleCode to modify data in the rowset during RowInit and RowInsert events without flagging the rows as changed. This ensures that only user changes cause the affected row to be saved.

   Note. Each rowset that is referenced by a grid or scroll area with the multi-row feature enabled should have the ChangeOnInit property for the rowset set to False. This includes lower-level rowsets. In addition, this property must be set prior to any RowInsert or RowInit PeopleCode for the affected row.

4. Empty rows at save.

   After a transaction is saved, any empty rows are discarded before the page is redisplayed to the user. An empty row means that the user did not access the data because PeopleCode or record defaults may have been used to initialize the row for the initial display.

   Note. PeopleCode save processing (SaveEdit and SavePreChange) PeopleCode executes for all rows in the buffer (including the empty ones). Therefore, SaveEdit and SavePreChange PeopleCode should be coded so that it is executed only if the field contains data, or if the row properties IsNew and IsChanged are both True. An alternative method is adding PeopleCode in the first save program in the component, to explicitly delete any row based on the IsNew and IsChanged properties. If you choose this method, then rows should be deleted from the bottom of the data buffer to the top (last row first).

See Also

Chapter 6, “PeopleCode and the Component Processor,” Deferred Processing Mode, page 113

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, “Rowset Class,” ChangeOnInit

Using the ImageReference Field

To associate an image definition with a field at runtime, the field has to be of type ImageReference. An example of this is referencing a red, yellow, or green light on a page, depending on the context.

To change the image value of an ImageReference field:

1. Create a field of type ImageReference.
2. Create the images you want to use.

   These images must be saved in PeopleSoft Application Designer as image definitions.
3. Add the field to a record that will be accessed by the page.
4. Add an image control to the page and associate the image control with the ImageReference field.
5. Assign the field value.

   Use the keyword **Image** to assign a value to the field. For example:

```
Local Record &MyRec;
Global Number &MyResult;

&MyRec = GetRecord();
If &MyResult Then
   &MyRec.MyImageField.Value = Image.THUMBSUP;
Else
   &MyRec.MyImageField.Value = Image.THUMBSDOWN;
End-If;
```

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer, "Creating Field Definitions"

Inserting Rows Using PeopleCode

When inserting rows using PeopleCode, you can either use the Insert method with a record object or create a SQL Insert statement using the SQL object. If you're doing a single insert, use the Record Insert method. If you're in a loop, and therefore calling the insert more than once, use the SQL object. The SQL object uses dedicated cursors, and if the database you're working with supports it, use bulk insert.

A *dedicated cursor* means that the SQL gets compiled only once on the database, so PeopleTools looks for the meta-SQL only once. This can increase performance.

For *bulk insert*, inserted rows are buffered and sent to the database server only when the buffer is full or a commit occurs. This reduces the number of round-trips to the database. Again, this can increase performance.

The following is an example of using the Record Insert method:

```
&REC = CreateRecord(Record.GREG);
&REC.DESCR.Value = "Y" | &I;
&REC.EMPLID.Value = &I;
&REC.Insert();
```

The following is an example using a SQL object to insert rows:

```
&SQL = CreateSQL("%INSERT(:1)");
&REC = CreateRecord(Record.GREG);
&SQL.BulkMode = True;
For &I = 1 to 10
   &REC.DESCR.Value = "Y" | &I;
   &REC.EMPLID.Value = &I;
   &SQL.Execute(&REC);
End-For;
```

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Record Class," Insert

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

Using OLE Functions

This section provides an overview of OLE functions and discusses how to:

- Use the Object data type.
- Share a single object instance.
- Use the Exec and WinExec functions.

Understanding OLE Functions

OLE automation is a Microsoft Windows protocol that enables one application to control another's operation. The applications communicate by means of an OLE object. One of the applications (called the automation server) makes available an OLE object that the second application (the client application) can use to send commands to the server application. The OLE object has methods associated with it, each of which corresponds to an action that the server application can perform. The client runs the methods, which cause the server application to perform the specified actions.

PeopleCode includes a set of functions that enable your PeopleCode program to be an OLE client. You can connect to any application that's registered as an OLE automation server and invoke its methods.

Note. Differences in Microsoft Windows applications from one release to the next (that is, properties becoming methods or vice versa) can cause problems with the ObjectGetProperty, ObjectSetProperty and ObjectDoMethod functions.

See the documentation for the OLE-automated application.

See Also

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

Using the Object Data Type

To support OLE, PeopleCode has a special data type, Object, which it uses for OLE objects. The purpose of the Object data type is to hold OLE objects during the course of a session so that you can run its methods. You cannot store Object data for any extended period of time.

Important! Object is a valid data type for variables, but not for record fields. Because OLE objects are by nature temporary, you cannot store Object data in a record field, including work record fields.

Some OLE object methods return data to the client. You can use such methods to get data from the automation server, if the method returns the data in a PeopleCode-supported data type. If the method returns data in an spreadsheet, for example, you cannot accept the data, because PeopleCode does not support spreadsheets.

Sharing a Single Object Instance

When you need the services of an OLE automation server, you create an instance of its OLE object, using the CreateObject function. After you have the object, you can run its methods as often as you like. You do not need to create a new instance of the object each time.

In a typical scenario, you have a PeopleSoft component that needs to access Microsoft Excel or Word, or some other automation server, perhaps one you have created yourself. Various PeopleCode programs associated with the component must run OLE object methods.

Rather than create a new instance of the OLE object in each PeopleCode program, you should create one instance of the OLE object in a PeopleCode program that runs when the component starts (such as RowInit) and assign it to a global variable. Then, any PeopleCode program can reference the object and invoke its methods.

Using the Exec and WinExec Functions

The WinExec and Exec built-in functions provide another way to start another application from PeopleCode. Unlike the OLE functions, however, Exec and WinExec do not enable you to control what actions the application takes after you start it. You can start the application, and if you use the synchronous option you can find out when it closes, but you cannot affect its course or receive any data in return.

WinExec is appropriate in two situations:

- When you want to start an application and continue processing.
- When you have a short, unvarying process that you want to run, such as copying a file.

The Exec function, unlike WinExec and the OLE functions, is not Microsoft Windows-specific. You can run it on an application server to call an executable on the application server platform, which in PeopleTools release 7 and later can be either Windows NT or UNIX.

Important! If you use the WinExec function with its synchronous option, the PeopleCode program (and the PeopleSoft application) remain paused until the called program is complete. If you start a program that waits for user input, such as Notepad, the application appears to hang until the user closes the called program. The synchronous option also imposes limits on the PeopleCode.

See Also

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

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

Using the Select and SelectNew Methods

This section provides an overview of the Select method and discusses how to use the Select method.

Understanding the Select and SelectNew Methods

The Select and SelectNew methods, like the ScrollSelect functions, enable you to control the process of selecting data into a page scroll area. The Select method selects rows from a table or view and adds the rows to either a rowset or a row. Let's call the record definition of the table or view that it selected from the *select record.* Let's call the primary database record of the top-level rowset object executing the method the *default scroll record.*

The select record can be the same as the default scroll record, or it can be a different record definition that has the same key fields as the default scroll record. If you define a select record that differs from the default scroll record, you can restrict the number of fields loaded into the buffers by including only the fields you actually need.

You can use these methods only with a rowset. A rowset can be thought of as a page scroll area.

A level zero rowset starts at the top level of the page, level zero, and contains all the data in the component buffers. A child rowset is contained by an upper-level rowset, also called the parent rowset. For example, a level one rowset could be considered the child rowset of a level zero, or parent, rowset. Or a level two rowset could be the child rowset of a level one rowset. The data contained in a child rowset depends on the row of the parent rowset.

When a rowset is selected into, any autoselected child rowsets are also read. The child rowsets are read using a Where clause that filters the rows according to the Where clause used for the parent rowset, using a Subselect.

The Select method automatically places child rowsets in the rowset object executing the method under the correct parent row. If it cannot match a child rowset to a parent row, an error occurs.

The Select method also accepts an optional SQL string that can contain a Where clause restricting the number of rows selected into the scroll area. The SQL string can also contain an Order By clause, enabling you to sort the rows.

The Select and SelectNew methods generate an SQL Select statement at runtime, based on the fields in the select record and the Where clause passed to them in the function call. This gives Select and SelectNew a significant advantage over the SQLExec function: they enable you to change the structure of the select record without affecting the PeopleCode program, unless the field affected is referred to in the Where clause string. This can make the application easier to maintain.

Also, if you use one of the meta-SQL constructs or shortcuts in the Where clause, such as %KeyEqual or %List, even if a field has changed, you do not have to change your code.

Unlike the ScrollSelect functions, neither Select or SelectNew allow you to operate in turbo mode.

Note. In addition to these methods, the SelectByKey record class method enables you to select into a record object. If you're only interested in selecting a single row of data, consider this method instead.

See Also

Chapter 5, "Accessing the Data Buffer," page 59

Using the Select Method

The syntax of the Select method is:

```
Select([parmlist], RECORD.selrecord [, wherestr, bindvars]);
```

Where *paramlist* is a list of child rowsets, given in the following form:

```
SCROLL.scrollname1 [SCROLL., scrollname2] . . .
```

The first *scrollname* must be a child rowset of the rowset object executing the method, the second *scrollname* must be a child of the first child, and so on.

This syntax does the following:

- Specifies an optional child rowset into which to read the selected rows.
- Specifies the select record from which to select rows.
- Passes a string containing a SQL Where clause to restrict the selection of rows or an Order By clause to sort the rows, or both.

Specifying Child Rowsets

The first part of the Select syntax specifies a child rowset into which rows are selected. This parameter is optional.

If you do not specify any child rowsets in *paramlist*, Select selects from a SQL table or view specified by *selrecord* into the rowset object executing the method. For example, suppose you've instantiated a level one rowset &BUS_EXPENSES_PER. The following would select into this rowset:

```
Local Rowset &BUS_EXPENSES_PER;

&BUS_EXPENSES_PER = GetRowset(SCROLL.BUS_EXPSNESE_PER);
&BUS_EXPENSES_PER.Select(RECORD.BUS_EXPENSE_VW,
"WHERE SETID = :1 and CUST_ID = :2", SETID, CUST_ID);
```

If the rowset executing the method is a level zero rowset, and you specify the Select method without specifying any child rowsets with *paramlist,*, the method reads only a single row, because only one row is allowed at level zero.

Note. For developers familiar with previous releases of PeopleCode: In this situation, the Select method is acting like the RowScrollSelect function.

If you specify a child rowset in *paramlist,* the Select method selects from a SQL table or view specified by *selrecord* into the child rowset specified in *paramlist*, under the appropriate row of the rowset executing the method.

In the following example, rows are selected into a child rowset BUS_EXPENSE_DTL, matching level-one keys, and with the charge amount equal to or exceeding 200, sorting by that amount:

```
Local Record &REC_EXP;
Local Rowset &BUS_EXPENSE_PER;

&REC_EXP = GetRecord(RECORD.BUSINESS_EXPENSE_PER;
&BUS_EXPENSE_PER = GetRowset(SCROLL.BUS_EXPSNESE_PER);
&BUS_EXPENSE_PER.Select(SCROLL.BUS_EXPENSE_DTL,
RECORD.BUS_EXPENSE_DTL, "WHERE %KeyEqual(:1) AND EXPENSE_AMT
>= 200 ORDER BY EXPENSE_AMT", &REC_EXP);
```

Specifying the Select Record

The record definition of the table or view being selected from is called the *select record*, and identified with RECORD.*selrecord.*. The select record can be the same as the primary database record associated with the rowset executing the method, or it can be a different record definition that has compatible fields.

The select record must be defined in PeopleSoft Application Designer and be a built SQL table or view (using Build, Project), unless the select record is the same record as the primary database record associated with the rowset.

The select record can contain fewer fields than the primary record associated with the rowset, although it must contain any key fields to maintain dependencies with other records.

If you define a select record that differs from the primary database record for the rowset, you can restrict the number of fields that are loaded into the buffers on the client work station by including only the fields you actually need.

The Where Clause

The Select method accepts a SQL string that can contain a Where clause restricting the number of rows selected into the object. The SQL string can also contain an Order By clause to sort the rows.

Select and SelectNew generate a SQL Select statement at runtime, based on the fields in the select record and the Where clause passed to them in the method parameters.

To avoid errors, the Where clause should explicitly select matching key fields on parent and child rows. You do this using the %KeyEqual meta-SQL.

Select Like RowScrollSelect

If the rowset executing the method is a level zero rowset, and you specify Select without specifying any child rowsets with *paramlist*, the method reads only a single row, because only one row is allowed at level zero.

Note. For developers familiar with previous releases of PeopleCode: In this situation, the Select method is acting like the RowScrollSelect function.

If you qualify the lower-level rowset so that it only returns one row, it acts like the RowScrollSelect method.

```
&RSLVL1 = GetRowset(SCROLL.PHYSICAL_INV);
&RSLVL2 = &RSLVL1(&PHYSICAL_ROW).GetRowset(SCROLL.PO_RECEIVED_INV);
&REC2 = &RSLVL2.PO_RECEIVED_INV;
If &PO_ROW = 0 Then
   &RSLVL2.Select(PO_RECEIVED_INV, "WHERE %KeyEqual(:1)
and qty_available > 0", &REC2);
End-if;
```

See Also

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

Using Standalone Rowsets

This section provides an overview of standalone rowsets and discusses how to:

- Use the Fill rowset method.
- Use the CopyTo rowset method.
- Add child rowsets.
- Use standalone rowsets to write a file.
- Use standalone rowsets to read a file.

Understanding Standalone Rowsets

Standalone rowsets are not associated with a component or page. Use them to work on data that is not associated with a component or page buffer. Prior to this release, this was done using derived work records. You still must build work pages.

Note. Standalone rowsets are not connected to the Component Processor, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets are not automatically applied at save time.

As with any PeopleTools object, the scope of standalone rowsets can be Local, Global, or Component. Consider the following code:

```
Local Rowset &MYRS;

&MYRS = CreateRowset(RECORD.SOMEREC);
```

This creates a rowset with SOMEREC as the level zero record. The rowset is unpopulated. Functionally, it is the same as an array of records.

Using the Fill Method

The Fill method fills the rowset by reading records from the database, by first flushing out all the contents of the rowset. A Where clause must be provided to get all the relevant rows.

```
Local Rowset &MYRS;
Local String &EMPLID;

&MYRS = CreateRowset(RECORD.SOMEREC);
&EMPLID = '8001';

&MYRS.Fill("where EMPLID = :1", &EMPLID);
```

Using the CopyTo Method

The CopyTo method copies like-named fields from a source rowset to a destination rowset. To perform the copy, it uses like-named records for matching, unless specified. It works on any rowset except the Application Engine state records. The following is an example:

```
Local Rowset &MYRS1, MYRS2;
Local String &EMPLID;

&MYRS1 = CreateRowset(RECORD.SOMEREC);
&MYRS2 = CreateRowset(RECORD.SOMEREC);

&EMPLID = '8001';

&MYRS1.Fill("where EMPLID = :1", &EMPLID);
&MYRS1.CopyTo(&MYRS2);
```

After running the previous code segment, &MYRS2 contains that same data as &MYRS1. Both &MYRS1 and &MYRS2 were built using like-named records.

To use the CopyTo method where there are no like-named records, you must specify the source and destination records. The following code copies only like-named fields:

```
Local Rowset &MYRS1, MYRS2;
Local String &EMPLID;

&MYRS1 = CreateRowset(RECORD.SOMEREC1);
&MYRS2 = CreateRowset(RECORD.SOMEREC2);

&EMPLID = '8001';

&MYRS1.Fill("where EMPLID = :1", &EMPLID);
&MYRS1.CopyTo(&MYRS2, RECORD.SOMEREC1, RECORD.SOMEREC2);
```

Adding Child Rowsets

The first parameter of the CreateRowset method determines the top-level structure. If you pass the name of the record as the first parameter, the rowset is based on a record. You can also base the structure on a different rowset. In the following example, &MYRS2 inherits the structure of &MYRS1:

```
Local Rowset &MYRS1, MYRS2;

&MYRS1 = CreateRowset(RECORD.SOMEREC1);
&MYRS2 = CreateRowset(&MYRS1);
```

To add a child rowset, suppose the following records describe a relationship. The structure is made up of three records:

- PERSONAL_DATA
- BUS_EXPENSE_PER
- BUS_EXPENSE_DTL

To build rowsets with child rowsets, use code like the following:

```
Local Rowset &rsBusExp, &rsBusExpPer, &rsBusExpDtl;
```

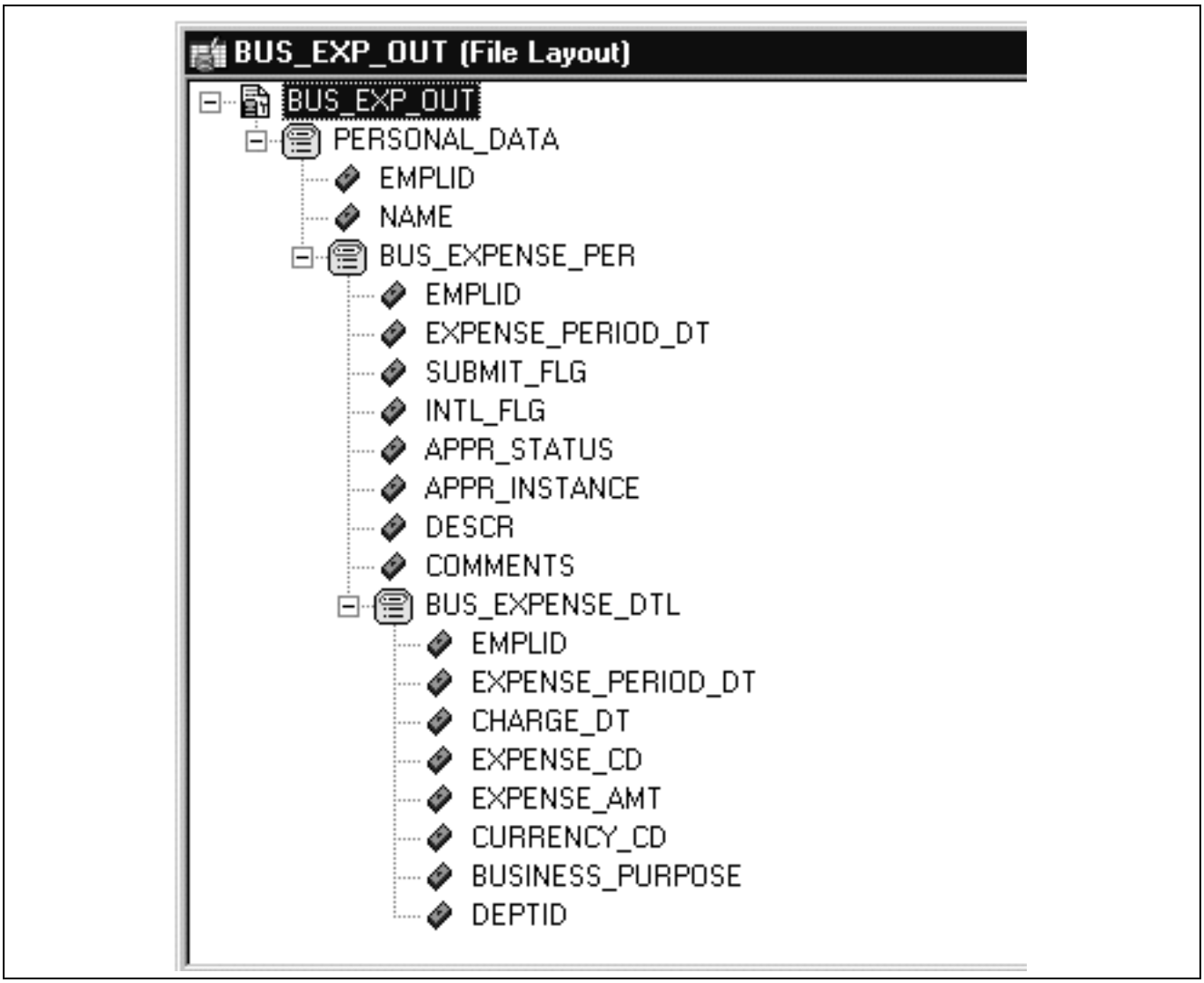
```
&rsBusExpDtl = CreateRowset(Record.BUS_EXPENSE_DTL);
&rsBusExpPer = CreateRowset(Record.BUS_EXPENSE_PER, &rsBusExpDtl);
&rsBusExp = CreateRowset(Record.PERSONAL_DATA, &rsBusExpPer);
```

Another variation is

```
&rsBusExp = CreateRowset(Record.PERSONAL_DATA,
CreateRowset(Record.BUS_EXPENSE_PER,
CreateRowset(Record.BUS_EXPENSE_DTL)));
```

Using Standalone Rowsets to Write a File

The following example shows the results of using standalone rowsets to write a file:



File layout example

The following example writes a file using a file layout that contains parent-child records:

```
Local File &MYFILE;
Local Rowset &rsBusExp, &rsBusExpPer, &rsBusExpDtl;
Local Record &rBusExp, &rBusExpPer, &rBusExpDtl;
```

```
Local SQL &SQL1, &SQL2, &SQL3;

&rBusExp = CreateRecord(Record.PERSONAL_DATA);
&rBusExpPer = CreateRecord(Record.BUS_EXPENSE_PER);
&rBusExpDtl = CreateRecord(Record.BUS_EXPENSE_DTL);

&rsBusExp = CreateRowset(Record.PERSONAL_DATA,
CreateRowset(Record.BUS_EXPENSE_PER,
CreateRowset(Record.BUS_EXPENSE_DTL)));
&rsBusExpPer = &rsBusExp.GetRow(1).GetRowset(1);

&MYFILE = GetFile("c:\temp\BUS_EXP.out", "W", %FilePath_Absolute);
&MYFILE.SetFileLayout(FileLayout.BUS_EXP_OUT);

&EMPLID = "8001";

&SQL1 = CreateSQL("%selectall(:1) where EMPLID = :2", &rBusExp, &EMPLID);
&SQL2 = CreateSQL("%selectall(:1) where EMPLID = :2", &rBusExpPer, &EMPLID);

While &SQL1.Fetch(&rBusExp)
   &rBusExp.CopyFieldsTo(&rsBusExp.GetRow(1).PERSONAL_DATA);
   &I = 1;
   While &SQL2.Fetch(&rBusExpPer)
      &rBusExpPer.CopyFieldsTo(&rsBusExpPer(&I).BUS_EXPENSE_PER);
      &J = 1;
      &SQL3 = CreateSQL("%selectall(:1) where EMPLID = :2
and EXPENSE_PERIOD_DT = :3", &rBusExpDtl, &EMPLID,
&rsBusExpPer(&I).BUS_EXPENSE_PER.EXPENSE_PERIOD_DT.Value);
      &rsBusExpDtl = &rsBusExpPer.GetRow(&I).GetRowset(1);
      While &SQL3.Fetch(&rBusExpDtl)
         &rBusExpDtl.CopyFieldsTo(&rsBusExpDtl(&J).BUS_EXPENSE_DTL);
         &rsBusExpDtl.InsertRow(&J);
         &J = &J + 1;
      End-While;

      &rsBusExpPer.InsertRow(&I);
      &I = &I + 1;
   End-While;
   &MYFILE.WriteRowset(&rsBusExp);
End-While;
&MYFILE.Close();
```

The previous code generates the following output file.

```
AA8001       Schumacher,Simon
BB8001       06/11/1989YNA0        Customer Go-Live Celebration
CC8001       06/11/198906/01/198908226.83             USDEntertain Clients  ⇒
 10100
BB8001       08/31/1989YNA0        Customer Focus Group Meeting
CC8001       08/31/198908/11/1989012401.58            USDCustomer Visit     ⇒
 10100
```

```
CC8001       08/31/198908/12/198904250.48              USDCustomer Visit     ⇒
 10100
CC8001       08/31/198908/12/198902498.34              USDCustomer Visit     ⇒
 10100
BB8001       03/01/1998YYP0        Attend Asia/Pacific Conference
CC8001       03/01/199802/15/1998011200                USDConference         ⇒
 00001
CC8001       03/01/199802/16/19980220000               JPYConference         ⇒
 00001
BB8001       05/29/1998NNP0        Annual Subscription
CC8001       05/29/199805/29/199814125.93              USDSoftware, Inc.     ⇒
 10100
BB8001       08/22/1998NNP0        Regional Users Group Meeting
CC8001       08/22/199808/22/19981045.69               USDDrive to Meeting   ⇒
 10100
CC8001       08/22/199808/22/19980912.44               USDCity Parking       ⇒
 10100
BB8001       12/12/1998NNP0        Customer Visit: Nevco
CC8001       12/12/199812/02/199801945.67              USDCustomer Feedback  ⇒
 00001
CC8001       12/12/199812/02/19981010.54               USDTo Airport         ⇒
 00001
CC8001       12/12/199812/03/19980610                  USDAirport Tax        ⇒
 00001
CC8001       12/12/199812/03/199804149.58              USDCustomer Feedback  ⇒
 00001
CC8001       12/12/199812/04/1998055.65                USDCheck Voicemail    ⇒
 00001
CC8001       12/12/199812/04/19980988                  USDAirport Parking    ⇒
 00001
CC8001       12/12/199812/04/199802246.95              USDCustomer Feedback  ⇒
 00001
CC8001       12/12/199812/04/199803135.69              USDCustomer Feedback   00001
```

Using Standalone Rowsets to Read a File

The following code shows an example of reading in a file and inserting the rows into the database:

```
Local File &MYFILE;
Local Rowset &rsBusExp, &rsBusExpPer, &rsBusExpDtl;
Local Record &rBusExp, &rBusExpPer, &rBusExpDtl;
Local SQL &SQL1;

&rBusExp = CreateRecord(Record.PERSONAL_DATA);
&rBusExpPer = CreateRecord(Record.BUS_EXPENSE_PER);
&rBusExpDtl = CreateRecord(Record.BUS_EXPENSE_DTL);

&rsBusExp = CreateRowset(Record.PERSONAL_DATA,
CreateRowset(Record.BUS_EXPENSE_PER,
CreateRowset(Record.BUS_EXPENSE_DTL)));
```

```
&MYFILE = GetFile("c:\temp\BUS_EXP.out", "R", %FilePath_Absolute);
&MYFILE.SetFileLayout(FileLayout.BUS_EXP_OUT);

&SQL1 = CreateSQL("%Insert(:1)");

&rsBusExp = &MYFILE.ReadRowset();
While &rsBusExp <> Null;
   &rsBusExp.GetRow(1).PERSONAL_DATA.CopyFieldsTo(&rBusExp);
   &rsBusExpPer = &rsBusExp.GetRow(1).GetRowset(1);
   For &I = 1 To &rsBusExpPer.ActiveRowCount
      &rsBusExpPer(&I).BUS_EXPENSE_PER.CopyFieldsTo(&rBusExpPer);
      &rBusExpPer.ExecuteEdits(%Edit_Required);
      If &rBusExpPer.IsEditError Then
         For &K = 1 To &rBusExpPer.FieldCount
            &MYFIELD = &rBusExpPer.GetField(&K);
            If &MYFIELD.EditError Then
               &MSGNUM = &MYFIELD.MessageNumber;
               &MSGSET = &MYFIELD.MessageSetNumber;
            End-If;
         End-For;
      Else
         &SQL1.Execute(&rBusExpPer);
         &rsBusExpDtl = &rsBusExpPer.GetRow(&I).GetRowset(1);
         For &J = 1 To &rsBusExpDtl.ActiveRowCount
            &rsBusExpDtl(&J).BUS_EXPENSE_DTL.CopyFieldsTo(&rBusExpDtl);
            &rBusExpDtl.ExecuteEdits(%Edit_Required);
            If &rBusExpDtl.IsEditError Then
               For &K = 1 To &rBusExpDtl.FieldCount
                  &MYFIELD = &rBusExpDtl.GetField(&K);
                  If &MYFIELD.EditError Then
                     &MSGNUM = &MYFIELD.MessageNumber;
                     &MSGSET = &MYFIELD.MessageSetNumber;
                  End-If;
               End-For;
            Else
               &SQL1.Execute(&rBusExpDtl);
            End-If;
         End-For;
      End-If;
   End-For;
   &rsBusExp = &MYFILE.ReadRowset();
End-While;
&MYFILE.Close();
```

Using Errors and Warnings

For the most part, errors and warnings display messages to users informing them about invalid data. For this reason, they are almost always placed in FieldEdit or SaveEdit PeopleCode, or in SearchSave PeopleCode for validation during search processing. In conjunction with edits, errors stop processing, while warnings allow processing to continue. When errors and warnings appear in places other than FieldEdit or SaveEdit, their effects vary.

This section discusses how to:

- Use errors and warning syntax.
- Use errors and warnings in edit events.
- Use errors and warnings in RowSelect events.
- Use errors and warnings in RowDelete events.
- Use errors and warnings in other events.

Using Error and Warning Syntax

Errors and warnings require only a message that the Component Processor displays to users. You can code the message into the error or warning statement, or you can use the message catalog. Use the message catalog with the MsgGet, MsgGetExplainText, and similar functions.

Errors and warnings use the same syntax. For example:

```
Error MsgGet(11100, 180, "Message not found.");
Warning MsgGet(11100, 180, "Message not found.");
```

Using Errors and Warnings in Edit Events

You can use the following PeopleCode events for validation edits: FieldEdit and SaveEdit. The Component Processor applies FieldEdit when the user changes a field, and SaveEdit when the user saves a component. Errors and warnings in these events display a message. Most errors and warnings appear in these event types, although you can use errors and warnings elsewhere.

FieldEdit Event Errors

You can use either the record field or component record field event. The record field event for each record runs before the component record field event for that record.

An error in FieldEdit prevents the system from accepting the new value of a field. The Component Processor highlights the problem field. The user must either change the field back to its original value or to something else which does not trigger the error. A warning enables the Component Processor to accept the new data. The Component Processor does not highlight a field that has warnings.

SaveEdit Event Errors

You can use the record field or the component record event. All record field events for a record run before the component record events.

An error in SaveEdit prevents the system from saving any row of data. The Component Processor does not update the database for any field if one field has an error. Although the Component Processor displays an error message, it does not turn any field red. Unlike FieldEdit errors, SaveEdit errors can happen anywhere on a page or component, for any row of data. The data causing the error may appear on a different page within the same group, or a row of data not currently displayed. If this is the case, the field in error is brought into view by the system.

A warning in SaveEdit also is applied to all data in the page or component, but the Component Processor will accept the data, if told to by the user. In a FieldEdit warning, the Component Processor displays a message box with the text and two buttons: OK and the standard Explain (the Explain button returns an explanation for the last message retrieved with the MsgGet function). In a SaveEdit warning, the message box contains an additional button, Cancel. OK accepts the data, overriding the warning and continuing the save process. Cancel ends the save process.

Because errors and warnings apply to all rows of data and all pages in a group, you must provide the user explicit information about what caused the error. Typically, you use the message catalog function to store messages and substitute variables into them. However, you can also facilitate this by concatenating in a field value. For example, if you have a stack of historical data on the page, you could use the following error statement:

```
Error ("The value exceeds the maximum on "|effdt|".");
```

Using Errors and Warnings in RowSelect Events

RowSelect PeopleCode filters out rows of data after the system applies search record criteria. It also can stop the Component Processor from reading additional rows of data.

Note. Errors and warnings should no longer be used in RowSelect processing; instead, use DiscardRow and StopFetching. The behavior of errors and warnings in RowSelect PeopleCode is retained for compatibility with previous releases of PeopleTools.

A warning causes the Component Processor to reject the current row, but the Component Processor continues reading more data. An error prevents more data coming into the page or component. The Component Processor accepts the row that causes the error, but does not read any more data. To reject the current row and stop loading additional rows, issue a warning and an error.

You must specify text for an error or warning, but the Component Processor does not display messages from RowSelect. You can still use the message text as a way of documenting the program.

See Also

Chapter 12, “Accessing PeopleCode and Events,” page 205

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” DiscardRow

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” StopFetching

Using Errors and Warnings in RowDelete Events

When you delete a row of data, the system prompts you to confirm. If you confirm, any record field RowDelete PeopleCode runs, and any component record RowDelete PeopleCode also runs. Errors and warnings in RowDelete display a message box.

A warning from RowDelete presents two choices: accept the RowDelete (the OK button), or cancel the RowDelete (the Cancel button). An error from RowDelete PeopleCode prevents the Component Processor from removing that row of data from the page.

Using Errors and Warnings in Other Events

Do not put errors or warning in PeopleCode attached to the FieldDefault, FieldFormula, RowInit, FieldChange, RowInsert, SavePreChange, WorkFlow, and SavePostChange events. These event types activate processing that a user has no direct control over. However, the Component Processor may issue its own errors and warnings when it runs PeopleCode and encounters an unrecoverable error. The Component Processor cancels the transaction to avoid unpredictable results.

See Also

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

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

Using the RemoteCall Feature

This section provides an overview of RemoteCall components and discusses how to:

- Decide between RemoteCall and PeopleSoft Process Scheduler.
- Modify PeopleSoft Process Scheduler programs to run with RemoteCall.

See Also

Chapter 8, "Using Methods and Built-In Functions," Think-Time Functions, page 136

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

Understanding RemoteCall Components

RemoteCall is a PeopleTools feature that enables executing a COBOL program remotely from within a PeopleSoft application. Remote calls are made using the RemoteCall PeopleCode function.

Because all PeopleCode runs on the application server, the RemoteCall PeopleCode function has more limited utility. However, RemoteCall can enable you to take advantage of existing COBOL processes.

The RemoteCall function is a synchronous call. The PeopleSoft system passes parameters to the remote program, and then waits while the program runs. When the remote program is done, it returns any results or status information to the client, which then resumes execution. This means that RemoteCall is a think-time function. RemoteCall is designed for fast response time, and has an API that provides programs with the response time needed for transaction processing. However, RemoteCall has no scheduling or multistep job capabilities. Each execution of RemoteCall is independent.

Note. For PeopleTools 8, you can no longer use RemoteCall to execute an Application Engine program. Use the CallAppEngine function instead.

The RemoteCall PeopleTools feature consists of the following components:

- PeopleCode program.

  This interface consists of the RemoteCall PeopleCode function. It is used from PeopleCode to start a remote program and process results. The PeopleCode program does not include any special code to specify where the remote program is executed. You can configure BEA Tuxedo to locally execute the program for testing.

- Remote program API.

  This is used by the remote COBOL program to receive or pass parameters and return status information.

- PeopleSoft RemoteCall service.

  The PeopleSoft application server, PSAPPSRV, advertises the RemoteCall service. The service receives requests from clients and starts the requested program. When the program is completed, it passes the parameters and status code back to the client.

- BEA Tuxedo.

  BEA Tuxedo is a message-based transaction monitor for distributed applications. No direct BEA Tuxedo calls need to be implemented in PeopleCode or remote programs.

PeopleCode Program

You can execute the RemoteCall function from PeopleCode associated with any Component Processor event except SavePostChange, SavePreChange, Workflow, RowSelect, or in any PeopleCode event resulting from a ScrollSelect or related function call. However, remote programs that change data should not be run as part of a SaveEdit process, because the remote program may complete successfully even though an error occurs later in the save process.

To call a remote program that changes data, use FieldChange PeopleCode in a record field associated with a command button, or from a pop-up menu item.

Do not use RemoteCall if you expect the remote program to return a large amount of data to the client, because data is passed back only through the parameters of the PeopleCode API.

Authorization to run a remote program is like authorization to run a PeopleCode program. Because a remote program is started from PeopleCode, the user has authorization to use the page that executes the PeopleCode.

The remote program runs in a different unit of work from the page. A commit is issued by PeopleTools if needed on the client before RemoteCall is called. This means that, by default, the remote program does not know about any database changes unless the page is saved before the program is called. After the remote program starts, it runs to completion and commits or ends before returning to the page. In this way, the remote program and the page do not have locking contention. To ensure that the save has actually been done, use the DoSaveNow built-in function.

When using RemoteCall to execute a COBOL program, two types of errors can occur:

- PeopleTools errors.

  An error could occur in PeopleTools or BEA Tuxedo, or the service might not be found. These are treated as hard errors by PeopleCode. An error message box appears, and that piece of PeopleCode is terminated. In the case of a PeopleTools error, the remote program always either returns a code of zero or terminates with a message due to a system error.

- Application-specific errors.

  Any error information specific to the remote application must be passed back in regular data variables, and the application can process these in an application-specific way. If you have a status code on which the application depends, you should initialize it to an invalid value to be sure the COBOL program does return the status code.

Because the remote program is executed synchronously, users receive an hourglass icon and cannot do anything in the current window until the remote application completes. They could move to another window and do processing there, or they could open another PeopleSoft window. They cannot cancel the remote program after it starts. If the program does not terminate in a timely fashion (as determined by the RemoteCall timeout set with PeopleSoft Configuration Manager), RemoteCall attempts to terminate the process and returns an error indicating that the program was terminated.

Remote Program API

The remote program API provides the functions to get and put data between the network and the COBOL program. These functions are implemented in C, but are callable from COBOL through the PTPNETRT program. For an example, see the PTPNTEST.CBL program.

Note. If these APIs are called when the program is not running as a remote program, ACTION-GET and ACTION-PUT return an error. All other actions return without doing anything.

If an unexpected error is found, call PTPNETRT with ACTION-RESET, then with ACTION-PUT to send back any error status variables, then with ACTION-DONE to send the buffer.

PeopleSoft RemoteCall Service

The RemoteCall service serves as a bridge between the PeopleCode API and remote COBOL programs. RemoteCall is one of many services advertised from the PSAPPSRV BEA Tuxedo server, and can be configured as part of the standard domain setup and administration.

The client sends the RemoteCall service request, consisting of the connect information and the program name, as well as any other parameters for the program, to the application server. The RemoteCall service then executes the program and passes it the connect string.

RemoteCall Programming Guidelines

Keep the following points in mind when using RemoteCall:

- Do not use RemoteCall for long-running batch jobs.

  As a general rule, if you think execution will take more than 15 seconds, you should not be using RemoteCall, but should instead use PeopleSoft Process Scheduler.

- RemoteCall is meant for running jobs on the server.

  It should not be used to invoke client-only programs. Support for local calling with RemoteCall is provided solely as a debugging and development aid. For client-only programs, use Declare Function, then call the external function from a library.

- If you do not want to modify an existing program, then pass only the program name and run control, and do not return any parameters.

  This way, the program requires few changes to run as a remote function.

Deciding Between RemoteCall and PeopleSoft Process Scheduler

COBOL application programs initiated by the RemoteCall service use the same COBOL application architecture used by PeopleSoft Process Scheduler. After being initiated by the dispatcher, COBOL application programs call the COBOL SQL API program, PTPSQLRT, to connect to the relational database management system to compile and execute SQL statements. You can design and implement COBOL programs to be understood by both PeopleSoft Process Scheduler and RemoteCall.

Follow these guidelines to select the optimal method for running a particular COBOL program:

- Use PeopleSoft Process Scheduler for asynchronous processes, or processes that can be scheduled, are multistep, or that require printed output.
- Use RemoteCall for synchronous processes that are quick (transaction processing types of processes).

Modifying PeopleSoft Process Scheduler Programs to Run with RemoteCall

To enable an existing program that runs under PeopleSoft Process Scheduler to run under RemoteCall as well, make the following changes:

- Include the PTCNETRT copy member.
- Include the PTCNCHEK member before the connection call to PTPSQLRT.
- Add the call to PTPNETRT ACTION-DONE just before the program terminates (after the call to disconnect from the database).

  This should be conditional on whether you are RUNNING-REMOTE-CALL.
- If you are running as a RemoteCall, ensure that PROCESS-INSTANCE OF PRUNSTATUS is not set.

  Otherwise your calls to PTCPSTAT try to update the PSPRCSRQST table. This does not cause an error, but it is unnecessary processing.

This program can now run from PeopleSoft Process Scheduler or from RemoteCall. If a program has to pass parameters, it must have RemoteCall-specific ACTION-GET and ACTION-PUT calls.

CHAPTER 9

Understanding HTML Trees and the GenerateHTML Function

This chapter discusses the GenerateTree function.

Using the GenerateTree Function

This section provides an overview of HTML tress and discusses how to:

- Build HTML tree pages.
- Use HTML tree rowset records.
- Use tree actions (events).
- Initialize HTML trees.
- Process events passed from a tree to an application.
- Add mouse-over ability to HTML trees.
- Add visual selection node indicators.

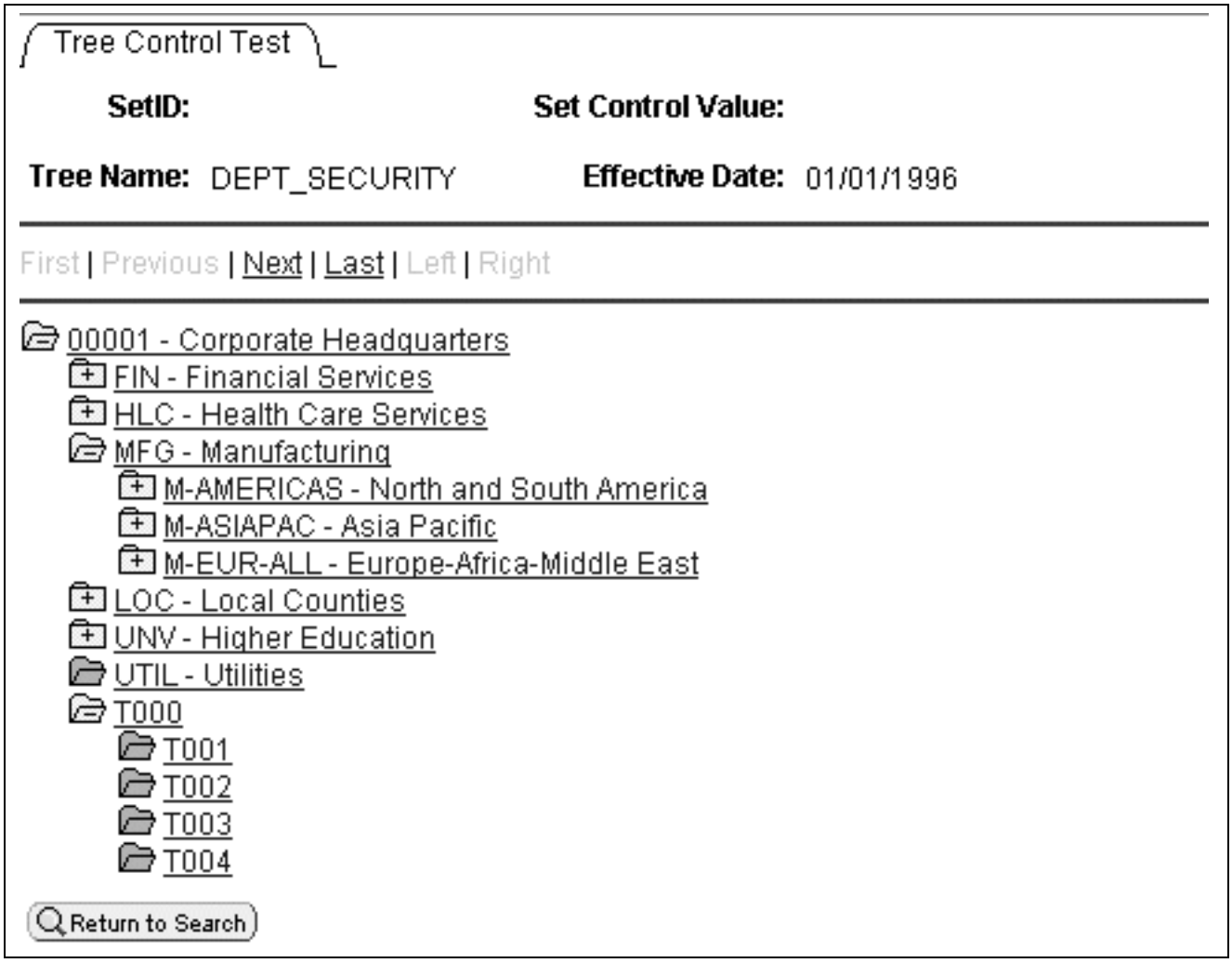
Understanding HTML Trees

Use the GenerateTree function to display data in a tree format. The result of the GenerateTree function is an HTML string, which can be displayed in an HTML area control. The tree generated by GenerateTree is called an HTML tree.

The GenerateTree function displays data from a rowset. You can populate this rowset using existing record data. You can also use the tree classes to display data from trees created using PeopleSoft Tree Manager.

To use this function, you must set up a page for displaying the data and populate a standalone rowset with the data to be displayed.

The following illustration shows an HTML tree:



HTML tree example

The positional links at the top of the page (First, Previous, Next, Last, Left, Right) enable the user to navigate around the tree. These links are automatically generated as part of the execution of GenerateTree.

When a node is collapsed, a plus sign appears on the node icon, and the node's children are hidden. When a node is expanded, all child nodes are displayed, and the icon has a minus sign. Icons without a plus or minus sign are terminal nodes, which have no children and cannot be expanded or collapsed.

Building HTML Tree Pages

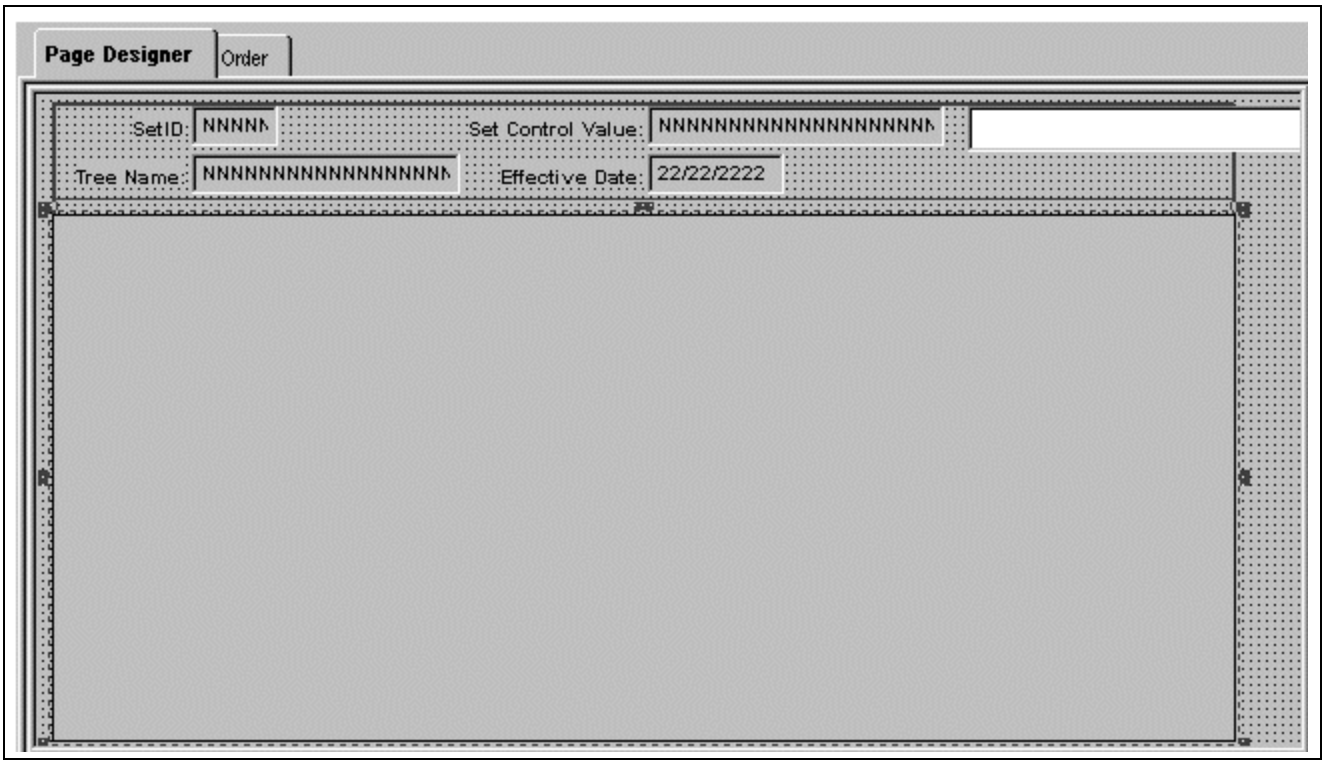
The page you use to display the HTML tree must contain the following:

- An HTML area used to display the HTML tree.
- A character field that has a page field name, is at least 46 characters long, and is invisible.

Note. The edit box should be invisible, but not display-only. An invisible edit box can't be seen by the user, but still has a buffer that can be written to. Page fields that have been specified as invisible do not need to be marked as Modifiable from HTML unless they are located on a page that is not active when GenerateTree is called. For example, if your application calls GenerateTree from one page, and then saves the result in a field that's displayed by an HTML area on another page in the component, the associated event field must be marked both Invisible and Modifiable from HTML.

Events are sent to the application from the HTML tree using the invisible field. The events are processed by FieldChange PeopleCode that is attached to the invisible field.

The following is an example page for an HTML tree:



Example PeopleSoft Application Designer HTML tree page

The large area that is selected in the example is the HTML area that displays the HTML tree. The HTML area is attached to the DERIVED_HTML.HTMLAREA field for this example.

The white edit box is the invisible field used to pass events from the HTML tree to the application and is attached to the DERIVED_HTML.TREECTLEVENT field for this example.

The edit box must have a page field name. In this example, the page field name is TREECTLEVENT.

Using HTML Tree Rowset Records

The GenerateTree function takes a prebuilt and populated rowset as a parameter. This rowset must have a certain structure and contain certain fields. In the examples below, it's a standalone rowset, that is, the rowset is created using the CreateRowset function. The fields necessary for the rowset are contained in the following record definitions:

- The header record TREECTL_HRD, containing the subrecord TREECTL_HDR_SBR.
- The node record TREECTL_NDE, containing the subrecord TREECTL_NDE_SBR.

The header record is the level zero record of the HTML tree rowset. It contains options for the HTML tree, such as the name of the collapsed node image, the height of the images, the number of pixels to indent each node, and so on.

The node record is the level one record of the HTML tree rowset. It contains the tree data and information about the data: the dynamic range leaf, the level, and so on.

There is a row in the level one scroll area for each node or leaf in the tree data.

To store additional application data with each node in the tree, you can incorporate the TREECTL_NDE_SBR into a record of your definition and use your record to define the HTML tree rowset.

For example, you might want to store application key values with each node record, so that when a user selects a node, you have the data you need to perform the action that you want.

The following are the relevant fields in TREECTL_HDR_SBR:

Field	Description
PAGE_NAME	Name of the page that contains the HTML area and the invisible field used to process the HTML tree events.
PAGE_FIELD_NAME	Page field name of the invisible field used to process the HTML tree events.
PAGE_SIZE	Number of nodes or leaves to send to the browser at a time. Set to 0 to send all visible nodes or leaves to the browser. The default value is 0.
DISPLAY_LEVELS	Number of levels to display on the browser at a time. The default value is 8.
COLLAPSED_IMAGE	Collapsed node image name. The default value is PT_TREE_COLLAPSED.
EXPANDED_IMAGE	Expanded node image name. The default value is PT_TREE_EXPANDED.
END_NODE_IMAGE	End node image name. The default value is PT_TREE_END_NODE.
LEAF_IMAGE	Leaf image name. The default value is PT_TREE_LEAF.
IMAGE_WIDTH	Image width in pixels. All four images need to be the same width. The default value is 15 pixels.
IMAGE_HEIGHT	Image height in pixels. All four images need to be the same height. The default value is 12 pixels.
INDENT_PIXELS	Number of pixels to indent each level. The default value is 20 pixels.
TREECTL_VERSION	Version of the HTML tree. The default value is 812. Used with the DESCR_IMAGE field in the TREECTL_HDR_SBR record.

The following are the relevant fields in TREECTL_NDE_SBR:

Field	Description
LEAF_FLAG	If this is a leaf, set to Y. The default value is N.
TREE_NODE	Node name.

Field	Description
DESCR	(Optional) Node description.
RANGE_FROM	The leaf's range from value.
RANGE_TO	The leaf's range to value.
DYNAMIC_FLAG	If this leaf has a dynamic range, set to Y. The default value is N.
ACTIVE_FLAG	Set to N for the node or leaf not to be a link. The default value is Y.
DISPLAY_OPTION	Set to N to display the name only. Set to D to display the description only. Set to B to display both the name and the description. Used for nodes only. The default value is B.
STYLECLASSNAME	Used to control the style of the link associated with the node or leaf. The default value is PSHYPERLINK.
PARENT_FLAG	If this node is a parent and its direct children are loaded now, set to Y. If this node is a parent and its direct children are loaded on demand, set to X. If this node is not a parent, set to N. The default value is N.
TREE_LEVEL_NUM	Set to the node's level. The default value is 1.
LEVEL_OFFSET	If a child node is to be displayed more than one level to the right of its parent, specify the number of additional levels. The default value is 0.
DESCR_IMAGE	Use to display an image after the node or leaf image and before the name or description. There is a space between the two images. The new image is not scaled. This field takes a string value, the name of an image definition created in PeopleSoft Application Designer. This field is only recognized if the TREECTL_VERSION field is greater than or equal to 812.
EXPANDED_FLAG	When a node's EXPANDED_FLAG is set to Y, the GenerateTree function expects the node's immediate children to be loaded into the &TREECTL rowset (such as in PostBuild), and GenerateTree generates HTML such that the node is expanded and its immediate children are displayed.

See Also

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

Using HTML Tree Actions (Events)

The GenerateTree function works with an HTML area control and an invisible field. When a user selects a node, expands a node, collapses a node, or uses one of the navigation links, that event (user action) is passed to the invisible field, and the invisible field's FieldChange PeopleCode is executed.

The FieldChange PeopleCode example program (below) checks for expanding (or collapsing) a node and selecting a node, by checking the first character in the invisible field. The following example checks for whether a node is selected:

```
If Left(TREECTLEVENT, 1) = "S" Then
```

In your application, you can check for the following user actions:

Event	Description
T*n*	Expand or collapse the node, whichever is the opposite of the previous state. *N* is the node's row number in the TREECTL_NODE rowset.
X*n*	Expand the node, but first load the children. The children are loaded in PeopleCode, then the event is passed to GenerateTree, so that the HTML can be generated with the node expanded. *N* is the node's row number in the TREECTL_NODE rowset.
F	Display the first page.
P	Display the previous page.
N	Display the next page.
L	Display the last page.
Q	Move the display left one level.
R	Move the display right one level.
S*n*	Select the node or leaf. *N* is the node's or leaf's row number in the TREECTL_NODE rowset.

Note. Drag-and-drop is not supported in an HTML tree.

Initializing HTML Trees

The PeopleCode for initializing the HTML tree for this example was put into the PostBuild event of the component that contained the page with the HTML area used with the HTML tree.

The PostBuild PeopleCode Example program is an example of how to initialize the HTML tree using the Tree classes and load only the root node into the HTML tree rowset.

The first time a user expands a node, the node's direct children are loaded into the HTML tree rowset by the FieldChange PeopleCode Example program, shown in the following section. This chunking functionality enables the HTML tree to support trees of any size with good performance.

You can't just copy either the PostBuild or FieldChange PeopleCode example programs into your application. You must modify them to make them work with your data. Changes that you must make to the PostBuild PeopleCode are as follows.

To modify the FieldChange PeopleCode to initialize HTML trees:

1. Set the PAGE_NAME and PAGE_FIELD_NAME fields.

   The PAGE_NAME field contains the name of the page that contains the HTML area and the invisible field that processes HTML tree events. The PAGE_FIELD_NAME field is the page field name of the invisible field that is used to process the HTML tree events.

   Note. This is the page field name of the invisible field, not the invisible field name.

2. Set tree-specific variables.

   The &SET_ID, &USERKEYVALUE, &TREE_NAME, &TREE_DT, and &BRANCH_NAME variables contain specific information about the tree. Set these values to the tree you want to open. In the example PeopleCode that follows, these varaibles are set as follows:

```
&SET_ID = PSTREEDEFN_VW.SETID;
&USERKEYVALUE = "";
&TREE_NAME = PSTREEDEFN_VW.TREE_NAME;
&TREE_DT = PSTREEDEFN_VW.EFFDT;
&BRANCH_NAME = "";
```

3. Set the PAGE_SIZE field.

   If you do not want the page to expand vertically to display the tree, set the PAGE_SIZE to a number of rows that will fit inside the HTML area. If some vertical expansion is okay, but you do not want the page to get too large, set the PAGE_SIZE to whatever value you like. Set the PAGE_SIZE to 0 if you don't care how big the page gets.

4. Set the DISPLAY_LEVELS field to the number of levels that will fit inside the HTML area.

   If this field is set too large, wrapping may occur. Positional links at the top of the HTML area enable the user to navigate as the tree expands.

5. (Optional) Set the DISPLAY_OPTION field.

   The default for the DISPLAY_OPTION field is to display both the node name and the description. You can display just the node name or just the description. The values for this field are:

Field Value	Description
N	Display the name only.

Field Value	Description
D	Display the description only.
B	Display both the name and description.

6. (Optional) Set the STYLECLASSNAME field for the root node.

   The STYLECLASSNAME field controls the style of the link associated with a node or leaf. The default for the STYLECLASSNAME is PSHYPERLINK. If PSHYPERLINK is not the style you want to use, change this field value to the style you want.

7. Change the last line to assign the output of GenerateTree to the field attached to the HTML area that will display the tree.

   In the example that follows, the HTML area control is the DERIVED_HTML.HTMLAREA. You must specify the record and field name associated with the HTML area control on your page.

PostBuild PeopleCode Example

The PeopleCode for initializing the HTML tree for this example was put into the PostBuild event of the component that contained the page with the HTML area used with the HTML tree.

This example shows how to initialize the HTML tree using the tree classes and load only the root node into the HTML tree rowset.

```
Component Rowset &TREECTL;

&NODE_ROWSET = CreateRowset(Record.TREECTL_NODE);
&TREECTL = CreateRowset(Record.TREECTL_HDR, &NODE_ROWSET);

&TREECTL.InsertRow(1);
&REC = &TREECTL.GetRow(2).GetRecord(1);

/* Set the HDR options:

1) PAGE_NAME - Name of the page that contains the HTML Area
and the invisible field that will be used to process the HTML
tree events.
2) PAGE_FIELD_NAME - Page field name of the invisible field that
will be used to process the HTML tree events.
3) PAGE_SIZE - Number of nodes or leaves to send to the browser at
a time.
Set to 0 to send all of the visible nodes or leaves to the browser.
Default value: 0
4) DISPLAY_LEVELS - Number of levels to display on the browser at
a time.  Default value: 8
5) COLLAPSED_IMAGE - Collapsed node image name.
Default value: PT_TREE_COLLAPSED
6) EXPANDED_IMAGE - Expanded node image name.
Default value: PT_TREE_EXPANDED
7) END_NODE_IMAGE - End node image name.
```

```
Default value: PT_TREE_END_NODE
8) LEAF_IMAGE - Leaf image name.  Default value: PT_TREE_LEAF
9) IMAGE_WIDTH - Image width.
All four images need to be the same size.  Default value: 15
10) IMAGE_HEIGHT - Image height.  Default value: 12
11) INDENT_PIXELS - Number of pixels to indent each level.
Default value: 20
*/
&REC.GetField(Field.PAGE_NAME).Value = "TREECTL_TEST";
&REC.GetField(Field.PAGE_FIELD_NAME).Value = "TREECTLEVENT";
&REC.GetField(Field.PAGE_SIZE).Value = 15;
&REC.GetField(Field.DISPLAY_LEVELS).Value = 8;
&REC.GetField(Field.COLLAPSED_IMAGE).Value = "PT_TREE_COLLAPSED";
&REC.GetField(Field.EXPANDED_IMAGE).Value = "PT_TREE_EXPANDED";
&REC.GetField(Field.END_NODE_IMAGE).Value = "PT_TREE_END_NODE";
&REC.GetField(Field.LEAF_IMAGE).Value = "PT_TREE_LEAF";
&REC.GetField(Field.IMAGE_WIDTH).Value = 15;
&REC.GetField(Field.IMAGE_HEIGHT).Value = 12;
&REC.GetField(Field.INDENT_PIXELS).Value = 20;

&SET_ID = PSTREEDEFN_VW.SETID;
&USERKEYVALUE = "";
&TREE_NAME = PSTREEDEFN_VW.TREE_NAME;
&TREE_DT = PSTREEDEFN_VW.EFFDT;
&BRANCH_NAME = "";

&MYSESSION = %Session;
&SRC_TREE = &MYSESSION.GetTree();
&RES = &SRC_TREE.OPEN(&SET_ID, &USERKEYVALUE, &TREE_NAME,
&TREE_DT, &BRANCH_NAME, False);

/* Just insert the root node into the &TREECTL Rowset.
If the root node has children, set the &PARENT_FLAG to 'X',
so that its children will be loaded on demand. */

&ROOT_NODE = &SRC_TREE.FindRoot();

If &ROOT_NODE.HasChildren Then
   &PARENT_FLAG = "X";
Else
   &PARENT_FLAG = "N";
End-If;

&NODE_ROWSET = &TREECTL.GetRow(2).GetRowset(1);
&NODE_ROWSET.InsertRow(1);
&REC = &NODE_ROWSET.GetRow(2).GetRecord(1);

/* Set the NODE values:

1) LEAF_FLAG - If this is a leaf set to "Y".  Default value: N
```

```
2) TREE_NODE - Node name.
3) DESCR - Node description.  (optional)
4) RANGE_FROM - Leaf's range from value.
5) RANGE_TO - Leaf's range to value.
6) DYNAMIC_FLAG - If this leaf has a dynamic range, set to "Y".
Default value: N
7) ACTIVE_FLAG - Set to "N" for the node or leaf not to be a link.
  Default value: Y
8) DISPLAY_OPTION - Set to "N" to display the name only.
Set to "D" to display the description only.
Set to "B" to display both the name and the description.
Only used for nodes.  Default value: B
9) STYLECLASSNAME - Used to control the style of the link
associated with the node or leaf.  Default value: PSHYPERLINK
10) PARENT_FLAG - If this node is a parent and its direct
children will be loaded now, set to "Y".  If this node is a
parent and its direct children are to be loaded on demand,
set to "X".  Default value: N
11) TREE_LEVEL_NUM - Set to the node's level.  Default value: 1
12) LEVEL_OFFSET - If a child node is to be displayed more than
one level to the right of its parent, specify the number of
additional levels.  Default value: 0
*/
&REC.GetField(Field.LEAF_FLAG).Value = "N";
&REC.GetField(Field.TREE_NODE).Value = &ROOT_NODE.NAME;
&REC.GetField(Field.DESCR).Value = &ROOT_NODE.DESCRIPTION;
&REC.GetField(Field.RANGE_FROM).Value = "";
&REC.GetField(Field.RANGE_TO).Value = "";
&REC.GetField(Field.DYNAMIC_FLAG).Value = "N";
&REC.GetField(Field.ACTIVE_FLAG).Value = "Y";
&REC.GetField(Field.DISPLAY_OPTION).Value = "B";
&REC.GetField(Field.STYLECLASSNAME).Value = "PSHYPERLINK";
&REC.GetField(Field.PARENT_FLAG).Value = &PARENT_FLAG;
&REC.GetField(Field.TREE_LEVEL_NUM).Value = 1;
&REC.GetField(Field.LEVEL_OFFSET).Value = 0;

&SRC_TREE.Close();
DERIVED_HTML.HTMLAREA = GenerateTree(&TREECTL);
```

Processing Events Passed from a Tree to an Application

Use the following FieldChange PeopleCode to process the events passed from an HTML tree to an application. The code that processes the load children event loads the direct children of a node the first time the node is expanded by the user. Changes that you must make to the FieldChange PeopleCode are as follows.

To modify the FieldChange PeopleCode:

1. Globally change TREECTLEVENT to the name of the invisible field used to process the events.
2. Set the tree-specific variables.

The &SET_ID, &USERKEYVALUE, &TREE_NAME, &TREE_DT and &BRANCH_NAME variables contain specific information about the tree. Set these values to the tree you want to open. In the example PeopleCode that follows, they are set like this:

```
&SET_ID = PSTREEDEFN_VW.SETID;
&USERKEYVALUE = "";
&TREE_NAME = PSTREEDEFN_VW.TREE_NAME;
&TREE_DT = PSTREEDEFN_VW.EFFDT;
&BRANCH_NAME = "";
```

3. (Optional) Set the DISPLAY_OPTION field.

   The default for the DISPLAY_OPTION field is to display both the node name and the description. You can display just the node name or just the description. The values for this field are:

Field Value	Description
N	Display the name only.
D	Display the description only.
B	Display both the name and description.

4. (Optional) Set the STYLECLASSNAME field for the root node.

   The STYLECLASSNAME field controls the style of the link associated with a node or leaf. The default for the STYLECLASSNAME is PSHYPERLINK. If PSHYPERLINK is not the style you want to use, change this field value to the style you want.

5. Change the assignment of the output of every GenerateTree call to the field attached to the HTML area that will display the tree.

   In this example, the HTML area control is the DERIVED_HTML.HTMLAREA. You must specify the record and field name associated with the HTML area control on your page.

6. Change the code that processes the select event to perform the action you want when the user selects a node or leaf.

   This section is marked as Process Select Event in the following code sample.

FieldChange PeopleCode Example

The following is the PostBuild PeopleCode example:

```
Component Rowset &TREECTL;

/* process load children event */
If Left(TREECTLEVENT, 1) = "X" Then
   &ROW = Value(Right(TREECTLEVENT, Len(TREECTLEVENT) - 1)) + 1;
   &NODE_ROWSET = &TREECTL.GetRow(2).GetRowset(1);
   &PARENT_REC = &NODE_ROWSET.GetRow(&ROW).GetRecord(1);
   &PARENT_LEVEL = &PARENT_REC.GetField(Field.TREE_LEVEL_NUM).Value;
   &ROW = &ROW + 1;
```

```
   &SET_ID = PSTREEDEFN_VW.SETID;
   &USERKEYVALUE = "";
   &TREE_NAME = PSTREEDEFN_VW.TREE_NAME;
   &TREE_DT = PSTREEDEFN_VW.EFFDT;
   &BRANCH_NAME = "";

   &MYSESSION = %Session;
   &SRC_TREE = &MYSESSION.GetTree();
   &RES = &SRC_TREE.OPEN(&SET_ID, &USERKEYVALUE, &TREE_NAME,
&TREE_DT, &BRANCH_NAME, False);

   /* Find the parent node and expand the tree one level below
the parent.  Insert just the direct children of the parent node
into the &TREECTL Rowset.  If any of the child nodes have
children, set their PARENT_FLAG to 'X', so that their children
are loaded on demand. */

   &PARENT_NODE = &SRC_TREE.FindNode(&PARENT_REC.
GetField(Field.TREE_NODE).Value, "");
   If &PARENT_NODE.HasChildren Then
      &PARENT_NODE.Expand(2);

      If &PARENT_NODE.HasChildLeaves Then
         /* Load the child leaves into the &TREECTL Rowset. */
         &FIRST = True;
         &CHILD_LEAF = &PARENT_NODE.FirstChildLeaf;
         While &FIRST Or
               &CHILD_LEAF.HasNextSib
            If &FIRST Then
               &FIRST = False;
            Else
               &CHILD_LEAF = &CHILD_LEAF.NextSib;
            End-If;
            If &CHILD_LEAF.Dynamic = True Then
               &RANGE_FROM = "";
               &RANGE_TO = "";
               &DYNAMIC_RANGE = "Y";
            Else
               &RANGE_FROM = &CHILD_LEAF.RangeFrom;
               &RANGE_TO = &CHILD_LEAF.RangeTo;
               &DYNAMIC_RANGE = "N";
            End-If;

            &NODE_ROWSET.InsertRow(&ROW - 1);
            &REC = &NODE_ROWSET.GetRow(&ROW).GetRecord(1);

            /* Set the NODE values:

1) LEAF_FLAG - If this is a leaf set to "Y".  Default value: N
2) TREE_NODE - Node name.
```

```
3) DESCR - Node description.  (optional)
4) RANGE_FROM - Leaf's range from value.
5) RANGE_TO - Leaf's range to value.
6) DYNAMIC_FLAG - If this leaf has a dynamic range, set to "Y".
Default value: N
7) ACTIVE_FLAG - Set to "N" for the node or leaf not to be a link.
  Default value: Y
8) DISPLAY_OPTION - Set to "N" to display the name only.
Set to "D" to display the description only.
Set to "B" to display both the name and the description.
Only used for nodes.  Default value: B
9) STYLECLASSNAME - Used to control the style of the link
associated with the node or leaf.  Default value: PSHYPERLINK
10) PARENT_FLAG - If this node is a parent and its direct
children will be loaded now, set to "Y".  If this node is a
parent and its direct children are to be loaded on demand,
set to "X".  Default value: N
11) TREE_LEVEL_NUM - Set to the node's level.  Default value: 1
12) LEVEL_OFFSET - If a child node is to be displayed more than
one level to the right of its parent, specify the number of
additional levels.  Default value: 0
*/
            &REC.GetField(Field.LEAF_FLAG).Value = "Y";
            &REC.GetField(Field.TREE_NODE).Value = "";
            &REC.GetField(Field.DESCR).Value = "";
            &REC.GetField(Field.RANGE_FROM).Value = &RANGE_FROM;
            &REC.GetField(Field.RANGE_TO).Value = &RANGE_TO;
            &REC.GetField(Field.DYNAMIC_FLAG).Value =
&DYNAMIC_RANGE;
            &REC.GetField(Field.ACTIVE_FLAG).Value = "Y";
            &REC.GetField(Field.DISPLAY_OPTION).Value = "B";
            &REC.GetField(Field.STYLECLASSNAME).Value =
"PSHYPERLINK";
            /* Leaves never have children. */
            &REC.GetField(Field.PARENT_FLAG).Value = "N";
            &REC.GetField(Field.TREE_LEVEL_NUM).Value =
&PARENT_LEVEL + 1;
            &REC.GetField(Field.LEVEL_OFFSET).Value = 0;

            &ROW = &ROW + 1;
         End-While;
      End-If;

      If &PARENT_NODE.HasChildNodes Then
         /* Load the child nodes into the &TREECTL Rowset. */
         &FIRST = True;
         &CHILD_NODE = &PARENT_NODE.FirstChildNode;
         While &FIRST Or
               &CHILD_NODE.HasNextSib
            If &FIRST Then
```

```
               &FIRST = False;
            Else
               &CHILD_NODE = &CHILD_NODE.NextSib;
            End-If;
            If &CHILD_NODE.HasChildren Then
               &PARENT_FLAG = "X";
            Else
               &PARENT_FLAG = "N";
            End-If;

            /* If the tree uses strict levels, set the
&LEVEL_OFFSET to the number of levels that the child node is to
the right of its parent minus 1. */
            If &SRC_TREE.LevelUse = "S" Then
               &LEVEL_OFFSET = &CHILD_NODE.LevelNumber -
&PARENT_NODE.LevelNumber - 1;
            Else
               &LEVEL_OFFSET = 0;
            End-If;

            &NODE_ROWSET.InsertRow(&ROW - 1);
            &REC = &NODE_ROWSET.GetRow(&ROW).GetRecord(1);
            &REC.GetField(Field.LEAF_FLAG).Value = "N";
            &REC.GetField(Field.TREE_NODE).Value = &CHILD_NODE.Name;
            &REC.GetField(Field.DESCR).Value =
&CHILD_NODE.Description;
            &REC.GetField(Field.RANGE_FROM).Value = "";
            &REC.GetField(Field.RANGE_TO).Value = "";
            &REC.GetField(Field.DYNAMIC_FLAG).Value = "N";
            &REC.GetField(Field.ACTIVE_FLAG).Value = "Y";
            &REC.GetField(Field.DISPLAY_OPTION).Value = "B";
            &REC.GetField(Field.STYLECLASSNAME).Value =
"PSHYPERLINK";
            &REC.GetField(Field.PARENT_FLAG).Value = &PARENT_FLAG;
            &REC.GetField(Field.TREE_LEVEL_NUM).Value =
&PARENT_LEVEL + 1;
            &REC.GetField(Field.LEVEL_OFFSET).Value = &LEVEL_OFFSET;

            &ROW = &ROW + 1;
         End-While;
      End-If;

      /* change the parent's PARENT_FLAG from 'X' to 'Y' */
      &PARENT_REC.GetField(Field.PARENT_FLAG).Value = "Y";

      HTMLAREA = GenerateTree(&TREECTL, TREECTLEVENT);
   End-If;

   &SRC_TREE.Close();
Else
```

```
   /* Process select event. */

   /* As an example, just display the selected node name or
leaf range as a MessageBox. */

   If Left(TREECTLEVENT, 1) = "S" Then
      &ROW = Value(Right(TREECTLEVENT, Len(TREECTLEVENT) - 1)) + 1;
      &NODE_ROWSET = &TREECTL.GetRow(2).GetRowset(1);
      &REC = &NODE_ROWSET.GetRow(&ROW).GetRecord(1);
      If &REC.GetField(Field.LEAF_FLAG).Value = "N" Then
         MessageBox(0, "", 0, 0, "The selected node is %1.",
&REC.GetField(Field.TREE_NODE).Value);
      Else
         If &REC.GetField(Field.DYNAMIC_FLAG).Value = "N" Then
            If &REC.GetField(Field.RANGE_FROM).Value =
&REC.GetField(Field.RANGE_TO).Value Then
               &TEMP = "[" | &REC.GetField(Field.RANGE_FROM).
Value | "]";
            Else
               &TEMP = "[" | &REC.GetField(Field.RANGE_FROM).
Value | " - " | &REC.GetField(Field.RANGE_TO).Value | "]";
            End-If;
         Else
            &TEMP = "[ ]";
         End-If;
         MessageBox(0, "", 0, 0, "The selected leaf is %1.", &TEMP);
      End-If;
   Else
      /* process all other events */
      HTMLAREA = GenerateTree(&TREECTL, TREECTLEVENT);
   End-If;
End-If;

/* done processing the event, so clear it */
TREECTLEVENT = "";
```

See Also

Chapter 9, “Understanding HTML Trees and the GenerateHTML Function,” Using HTML Tree Actions (Events), page 170

Adding Mouse-Over Ability to HTML Trees

To add mouse-over ability to HTML tree elements, you must add a few fields to the TREECTL_HDR_SBR record, and some additional PeopleCode to the program, to set the values and set the images.

To add mouse-over ability to the HTML tree:

1. Add fields to the TREECTL_HDR_SBR record.

   Add the following fields to the tree control header record:

- COLLAPSED_MSGNUM
- COLLAPSED_MSGSET
- END_NODE_MSGNUM
- END_NODE_MSGSET
- EXPANDED_MSGNUM
- EXPANDED_MSGSET
- LEAF_NODE_MSGNUM
- LEAF_NODE_MSGSET

2. Add the following PeopleCode to set the message set and number for the mouse-over text.

```
&REC.GetField(Field.EXPANDED_MSGSET).Value = 2;
&REC.GetField(Field.EXPANDED_MSGNUM).Value = 903;
&REC.GetField(Field.COLLAPSED_MSGSET).Value = 2;
&REC.GetField(Field.COLLAPSED_MSGNUM).Value = 904;
&REC.GetField(Field.END_NODE_MSGSET).Value = 2;
&REC.GetField(Field.END_NODE_MSGNUM).Value = 905;
&REC.GetField(Field.LEAF_MSGSET).Value = 2;
&REC.GetField(Field.LEAF_MSGNUM).Value = 906;
```

3. Add fields to the TREECTL_NDE_SBR record.

   Add the following fields:

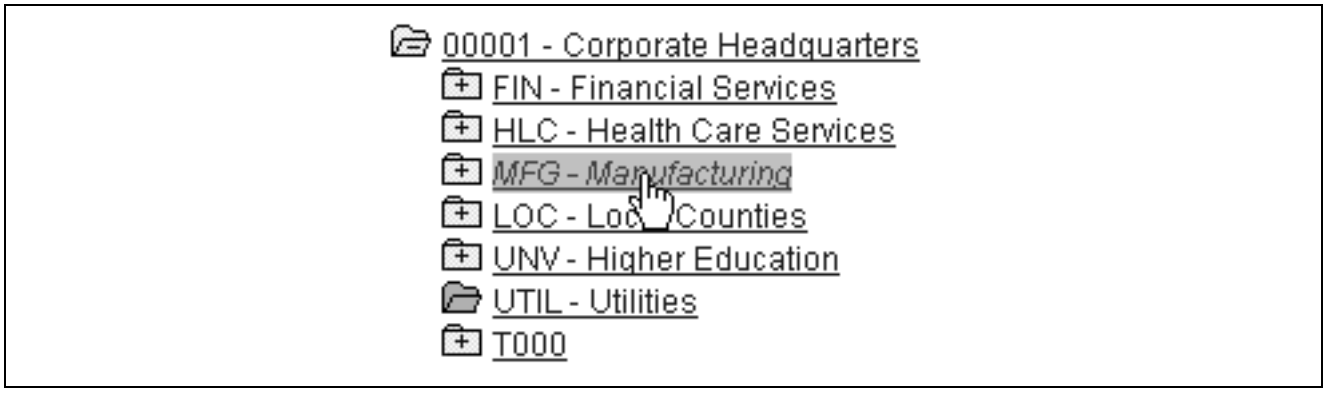
   - DESCR_MSGNUM
   - DESCR_MSGSET

4. Add PeopleCode to set the DESCR_MSGNUM and DESCR_MSGSET fields.

   These two fields should be set to the correct message number and message set values that contain the text to be used as the mouse-over text.

Adding Visual Selection Node Indicators

Sometimes users need a visual indicator to specify which node has been selected, such as a different color or style for the selected node, as shown in the following example:



Showing selected node style

To add selected node highlighting:

1. Add the field NODESELECTEDSTYLE to the TREECTL_HDR_SBR record.
2. Add PeopleCode to set the NODESELECTEDSTYLE field to provide the highlighting effect.

   The NODESELECTEDSTYLE field takes the name of a style class.

   The following example uses the PSTREENODESELECTED style.

```
&REC.GetField(Field.NODESELECTEDSTYLE).Value = "PSTREENODESELECTED";
```

   The style of the selected node can be set when processing the select event.

Note. You also must reset the style of the previous selected node when processing the select event. To find the previous selected node, you can search the node rowset looking for a node with a STYLECLASSNAME equal to the style you've set for selected nodes. Alternately, you could keep a global variable with the node's index in the rowset. However, if you keep an index variable, you may have to update the index when processing the load children event.

Specifying Override Images

You specify different images to represent the nodes in a tree by using the TREECTL_NODE record.

To specify override images:

1. Add the following fields to the tree control node record:
   - OVERRIDE_IMAGE
   - OVERRIDE_MSGSET
   - OVERRIDE_MSGNUM
2. Add PeopleCode to use the override values when writing tree control node records.

CHAPTER 10

Understanding File Attachments and PeopleCode

This chapter discuses using file attachments in applications and PeopleCode.

Using File Attachments in Applications

This section provides an overview of file attachment architecture and discusses how to:

- Debug file attachment problems.
- Configure multiple application servers to support file attachments.
- View file attachments.
- Use chunking with attachments.
- Use the Microsoft Windows NT 4 FTP server.
- Use non-DNS URLs with UNIX.
- Use attachment functions.
- Name URLs and file attachments.
- Convert user filenames.

Understanding File Attachment Architecture

File attachments are supported by using PeopleCode built-in functions that implement the transfer of a file to or from a browser using the application server. The file is either stored on, or retrieved from, a file server or database tables, referred to as the data storage system.

Files can be transferred back and forth from the browser to the web server, to the application server, to the data storage system.

The browser-web server transfer is performed using a standard HTML form construct. This can be done securely in an encrypted fashion if the web server uses Secure Sockets Layer (SSL) to communicate to the browser. After the file is received at the web server, 1 megabytes chunks are brought from the web server to the application server and stored to the database.

Note. The 1 megabyte transfer size is not customizable.

Once the entire file is transferred, the application server reassembles the file and sends it to either a FTP server or to a database table, depending. After the transfer is complete, the file is deleted from the database.

Note. The *MaxSize* variable is checked before the file is transferred to the web server. If the file exceeds the value specified with the *MaxSize* parameter, it is not transferred, and the system issues an error message.

The web server-application server transfer is performed by using BEA Jolt, which is securely encrypted. Because this transfer is done using the standard BEA Jolt mechanism, no additional settings to the firewall are required (there's no need to open additional ports). On the application server, the file is stored in one of the following temporary directories, depending on the operation:

- /tmp/PSFTP/A*TransactionNumber* (for attach operation)
- /tmp/PSFTP/V*TransactionNumber*(for view operation)

  TransactionNumber is randomly generated for both attaching and viewing.

By using separately named subdirectories for each attachment, support for the simultaneous transfer of identically named files is guaranteed, and attachments won't be mixed up.

The application server-data storage system transfer is performed either using File Transfer Protocol (FTP) or directly to a database table.

The FTP transfer can be securely encrypted. The FTP service is invoked on the application server by a script that is passed to the FTP binary. This script is in /tmp and is called ftp*Number*.txt. The output of invoking the script is called ftp*Number*.log (The number variable is a random number.) These files are left in /tmp for debugging, with the assumption that access to the application server is restricted and somewhat secure. For your own security, you may want to delete these files on a weekly basis.

Note. You cannot specify a different port for an attachment when using an FTP server. You can only use the default port of 21.

No log file or script is required for writing to database tables. For debugging, these commands leave traces in the PeopleCode trace files if Internal Tracing (256) is selected.

Note. The search API does not work with database attachments.

Debugging File Attachment Problems

To verify that the FTP transfer was successful, access the application server and examine the ftp*Number*.log file and the ftp*Number*.txt file.

The following is an example of a log file. It shows that the file transfer was successful.

```
pt-hp01:$ cat ftp1039.txt
open 216.131.219.51
user ANONYMOUS TEST
bin
lcd /
lcd tmp
lcd PSFTP
lcd V13333
cd /
get "IA_FA_Test.TXT" "IA_FA_Test.TXT"
quit
pt-hp01:$
pt-hp01:$ cat ftp1039.log
about to submit ftp -vin /tmp/ftp1039.txt
Connected to 216.131.219.51.
220 TCARREON052500 Microsoft FTP Service (Version 4.0).
331 Anonymous access allowed, send identity (e-mail name) as password.
```

```
230 Anonymous user logged in.
200 Type set to I.
Local directory now /
Local directory now /tmp
Local directory now /tmp/PSFTP
Local directory now /tmp/PSFTP/V13333
250 CWD command successful.
200 PORT command successful.
150 Opening BINARY mode data connection for IA_FA_Test.TXT(72 bytes).
226 Transfer complete.
72 bytes received in 0.06 seconds (1.21 Kbytes/s)
221
File Transfer Successful
pt-hp01:$
```

If the file transfer wasn't successful, error messages in the log can help you determine the problem.

A common reason that a transfer fails is that the FTP server was not accessible from the application server. This could be due to a wrong password or wrong account name, or perhaps because the application server was unable to resolve the FTP server hostname. Perhaps the FTP server was down and the application server was unable to route to it. Try pinging the FTP server from the application server system, and then try to FTP to the FTP server from the application server.

If you have a Microsoft Windows NT file server, it's possible that the hostname for the NT system is not associated with a fixed IP address and is not resolvable using the Domain Name System (DNS). If you have the application server on a UNIX machine (such as Solaris, HP-UX, or AIX), the application server can only resolve the hostname using DNS (or perhaps the Network Information System [NIS] or an /etc/hosts file), but *not* using WinBeui or WINS. This means that the application server won't be able to convert the hostname indicated for the Microsoft Windows NT file server into an IP address and route to it.

If you suspect this is the problem, try the following:

1. Telnet to the UNIX application server.
2. Try to ping the suspect hostname.

   For example:

   ```
   ping TCARREON052500
   ```

If this fails, you must resolve the problem by either specifying the IP address in the FTP URL or by putting your hostname into DNS, NIS, or a hosts file so that UNIX can resolve it.

Debugging Database Attachments

Turn on tracing for internal PeopleCode command using either the SetTracePC function with the value %TracePC_IntFuncs (256), or the trace functions used with PeopleSoft Pure Internet Architecture, to generate information about each step of the attachment process for database attachments.

Debugging Paths

In general, FTP servers require the full path from the top-level directory, rather than the path from the home directory (a relative path). If you've been able to add an attachment but can't view it, verify that you're using a full, absolute path, as opposed to a relative path.

Attachments with Double-Byte Character Filenames

To successfully upload attachments whose filename comes from a language which uses a double-byte character string, such as Japanese, from your web browser to either an FTP server or database table, the application server must be running in an environment that supports double-byte character languages.

If the destination for your attachment is an FTP server, that FTP server must also be running in an environment that supports double-byte character languages. (The web server can be running on either an English environment or a double-byte character language environment.)

Configuring Multiple Application Servers to Support File Attachments

When you have multiple application servers, the /tmp/PSFTP/DomainIDxxx directory is created by the first application server process to run on your application server. The assumption is that all such application server processes run with the same user ID. However, a single computer might run application servers with many user IDs.

The default /tmp/PSFTP is created so that all users can access it (777 permissions, or drwxrwxrwx). For proper security, however, the DomainIDxxx directory is created so that only that application server can access that directory (711 permissions, or drwx--x--x). If multiple users need read and write access, you must change the permissions.

When using file objects, the service request can be sent to different application servers, so that if a file was created under one application server machine environment and that service is next serviced on a different application server machine, the file might not exist. Therefore, your application should check that the file is open, particulary when returning from a user interraction (since the process of the handling user interface results in a serialization/deserialization of all the transaction states.)

Viewing File Attachments

When viewing a file attachment, the browser invokes a viewer based on the content-type Multipurpose Internet Mail Extensions (MIME) category sent in the response header from the web server.

For example, if the user tried to view an MP3 file, the response header sent to the browser by the web server would indicate the audio/MPEG content type.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Mon, 01 Oct 2001 21:25:51 GMT
Content-Type: audio/mpeg
Accept-Ranges: bytes
Last-Modified: Mon, 01 Oct 2001 21:00:26 GMT
ETag: "78e21918bc4ac11:cc8"
Content-Length: 60
```

Notice that the Content-Type is audio/mpeg. The browser uses this MIME type to determine that a viewer for audio/MPEG is appropriate. If the web server did not send this content-type header, the browser wouldn't be able to determine the nature of the file being transmitted, and would be unable to invoke the correct application. The browser would try to display the file as text/plain. This might be the wrong assumption.

The web server determines the MIME type by looking at the extension of the attachment file and mapping it to a MIME type. The mapping is done in the following ways, depending on the web server and platform:

- For Microsoft IIS, the mapping is done by using the standard Microsoft Windows extension association mechanism.
- For Apache, the mapping is done by editing a configuration file called mime.types.
- For WebLogic, a file called web.xml does the mapping.

The file web.xml is located in the WEB-INF subdirectory. This file contains a section that looks like this:

```
</mime-mapping>
 <mime-mapping>
   <extension>
     doc
   </extension>
   <mime-type>
      application/msword
   </mime-type>
 </mime-mapping>
 <mime-mapping>
   <extension>
     xls
   </extension>
   <mime-type>
      application/vnd.ms-excel
   </mime-type>
 </mime-mapping>
```

Let’s say you want to add an extension that causes log files to be interpreted as plain text files.

To determine the correct MIME type, check Request for Comments (RFC) documents 2045, 2046, 2047, 2048, and 2077, which discuss internet media types and the internet media type registry.

After checking the RFCs, you determine that the correct MIME type is text/plain. The following is an example of you would add to the above section of web.xml:

```
</mime-mapping>
 <mime-mapping>
   <extension>
     doc
   </extension>
   <mime-type>
      application/msword
   </mime-type>
 </mime-mapping>
 <mime-mapping>
   <extension>
     log
   </extension>
   <mime-type>
      text/plain
   </mime-type>
 </mime-mapping>
 <mime-mapping>
   <extension>
     xls
```

```
    </extension>
    <mime-type>
       application/vnd.ms-excel
    </mime-type>
  </mime-mapping>
```

Once you save the file, the log extension is now associated with the content type of text/plain.

Note. You must restart the WebLogic server before these changes are recognized.

Similar mappings can be achieved on other web servers, such as IBM WebSphere or Netscape Enterprise Server.

See your web server documentation.

Note. When trying to view the objects, the extension must match exactly what is set up in the web.xml file. This value is case-sensitive. If the object view appears garbled, chances are that either the extension is not set up in the web.xml file, or there is a case mismatch.

Using Chunking with Attachments

When writing data to database tables, the data is automatically chunked, or stored in different rows of the database table. The size of each chunk is determined by the Max Chunk File Size field on the PSOPTIONS page. The default value of this field is 28,000 kilobytes (K).

Because each file is chunked, you can't directly pull data from the table. You should use the provided attachment functions, which automatically put the data back together into one file for you. Because the chunk size is stored with the file, if you change the system chunk size you can still retrieve files with different chunk sizes.

DB2 UDB for OS/390 and z/OS Considerations

If you store file attachments on DB2 UDB for OS/390 and z/OS, you're limited to chunk sizes of 3862K for a 4K tablespace, or 32444K for a 32K page tablespace. By default, the only tablespace with 32K is PSIMAGE.

See Also

Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," PeopleTools Options

Using the Microsoft Windows NT 4 FTP Server

If you're using FTP, and running Microsoft Windows NT 4 with the standard Microsoft FTP server, you must take the following into consideration.

Non-Microsoft FTP server programs on Microsoft Windows NT, the Microsoft Windows 2000 FTP server, as well as all UNIX FTP servers, execute a chroot command to the home directory of the FTP server automatically.

However, the Windows NT 4 FTP server does not follow this behavior. When an FTP client connects to the Microsoft FTP server, the client can gain access to the entire drive, not just the FTP home directory.

For example, suppose you are running the Microsoft Windows NT 4 FTP server and the home directory is c:\inetpub\ftproot.

When you log in, your current directory is c:\inetpub\ftproot.

When you type the following:

```
cd \
```

Your current directory is now c:\.

If you are not running the Microsoft Windows NT 4 FTP server, and the home directory is the same: c:\inetpub\ftproot.

When you type the following:

```
cd \
```

Your current directory is now c:\inetpub\ftproot.

This means that you *must* know the FTP server associated with the URL.

If you are using a Windows NT 4 FTP server, or another FTP server that shows this behavior, your URLs should have the following format: ftp://ftp:ftp@NT4system/inetpub/ftproot/tmp. Note that the home directory of the FTP server must be appended to the address.

If you are not using these sorts of servers, your URLs should have the following format: ftp://ftp:ftp@NT4system/tmp.

Using Non-DNS URLs with UNIX

The URL passed to file attachments generally has the following format: ftp://user:pass@systemname/dir1/dir2.

This assumes that the domain name *systemName* can be resolved with DNS.

Some Microsoft Windows systems can resolve *systemName* even when it can't be resolved with DNS. However, some UNIX systems cannot these resolve names.

If you are using a UNIX system, and the domain name cannot be resolved with DNS, use the IP address. The following example assumes *systemName* has the IP address of 123.123.123.123: ftp://user:pass@123.123.123.123/dir1/dir2.

To determine the IP address, you can try to ping the system by typing the following:

```
ping systemName
```

You should use domain names that can be resolved. Only use IP addresses when absolutely necessary.

Using Attachment Functions

To store attachments in a database, include the FILE_ATTACH_SBR subrecord and the FILE_ATTACH_WRK work record in your component when using the file attachment functions.

Note. You do not have to include the FILE_ATTACH_SBR subrecord in your component; you could just include the fields. However, it's better to include the subrecord.

The FILE_ATTACH_SBR subrecord contains the following fields:

Field	Description
ATTACHSYSFILENAME	The system file name.
ATTACHUSERFILE	The user file name.

No PeopleCode is associated with this subrecord. Developers should include this subrecord in target records for using attachments.

The FILE_ATTACH_WRK work record contains the following fields:

Field	Description
ATTACHADD	Contains a PeopleCode program used for adding attachments (the AddAttachment built-in function).
ATTACHDELETE	Contains a PeopleCode program used for deleting attachments (the DeleteAttachment built-in function).
ATTACHVIEW	Contains a PeopleCode program used for viewing attachments (the ViewAttachment built-in function).

The PeopleCode program supplied by PeopleSoft with this record is associated with the FieldChange events for each function, and the RowInit event for the ATTACHADD function. Because the PeopleCode is in the form of functions, this work record acts as a container for the appropriate methods related to attachments. Instead of including this work record in your component, you can use your own work record and call these functions as needed from within your work record.

For storing attachments in the database, you must create a record that receives the attachments. This record can be called anything, but it must have the FILE_ATTDET_SBR subrecord included in it, and have no additional fields in it. The subrecord FILE_ATTDET_SBR has the following fields:

Field	Description
ATTACHSYSFILENAME	A unique system file name.
FILE_SEQ	The file sequence number (used in chunking).
VERSION	Version number.
FILE_SIZE	The physical size of the file.
LASTUPDDTTM	Last update date and time.
LASTUPDOPRID	The employee ID of the last user to update the attachment.
FILE_DATA	The data of the file.

To use the attachment functions:

1. Insert the subrecord FILE_ATTACH_SBR into a record used with your page.

   Typically, this record is inserted as part of the primary database record.

2. To store the file in the database, insert the subrecord FILE_ATTDET_SBR into a record used with your page.

   This record can be called anything, but it must have the FILE_ATTDET_SBR subrecord included in it, and have no additional fields in it. This is the record that you point to with the URL when specifying storing a file in a database table.

3. Add buttons to add, delete, and view attachments as needed.

You can either use the FILE_ATTACH_WRK record for the necessary record fields, or you can use your own work fields.

4. Set up the URL.

   If you're using the FTP archive, suppose the system is fileserver.ps.com, the user account is *joe*, and the password is *secret*. The full URL is: ftp://joe:secret@fileserver.ps.com/.

   You can either create an entry for this URL in the URL Maintenance page, or specify it in your PeopleCode.

   If you specify the URL in your PeopleCode, you don't have to hard-code the username and password. You can dereference them using a variable or record field. This enables you to use encryption with the username and password.

   If you're copying to a database table, you can either create an entry for this record (URL) in the URL Maintenance page, or specify it in your PeopleCode, as follows:

   ```
   record://RESUMES_DB
   ```

5. Modify the PeopleCode to support your application.

   If you use the supplied PeopleCode programs, you must specify the URL. You may also want to modify the extension of the expected files (the default is "", which is any file, or *.* You may want to indicate ".doc", ".xl", "*.html", "*.exe", and "*.tar".). Read the comments in ATTACHADD FieldChange program for instructions.

   To use the FILE_ATTACH_WRK record for buttons, you may want to use component PeopleCode for your application-specific code. This enables you to keep the code in FILE_ATTACH_WRK generic. If you use your own work fields, you can write your own code and call the functions in FILE_ATTACH_WRK as needed.

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

Coding Considerations Using the Attachment Functions

All file attachments are performed using PeopleCode built-in functions, such as AddAttachment, ViewAttachment, GetAttachment, and so on. These functions move a file into and out of a file storage system, either the database or a file system. The file system movement is done using the FTP protocol.

Because these functions abstract the storage of the attachments, you can use either database or FTP file systems. The system to be used is determined by the URL passed as the first parameter in the attachment built-in function. The actual value of the URL is maintained on the URL Maintenance page. This means that the you don't know whether users will store attachments in a database or a file system.

The following PeopleCode functions are used with administration work:

- DeleteAttachment deletes unwanted attachments.

  There is no roll-back ability. For example, suppose a user selects to view an attachment, then cancels before the attachment is displayed. A copy of the attachment may still be in the user's temp directory.

- CopyAttachments moves all of the attachments specified from one file storage system (database or FTP) to another (database or FTP).

- CleanAttachments is an audit-like script that removes attachments that exist in the database that are not referenced by any record field.

If you only have a few files to copy using CopyAttachments, you may want to use the CopyFiles page (part of the PeopleTools Administration pages). The CleanAttachments function is also available from this page.

To schedule a regular job to clean up orphaned file attachments, you can use the Application Engine program CLEANATT84.

Store all references to attachments using the standard ATTACHSYSFILENAME field. Do not reuse this field to store incomplete or nonstandard versions of the name. For example, you should not store the full URL version of the attachment in this column and then use PeopleCode to parse the URL before invoking attachment commands. While this may work for get or put attachment calls, the Clean Attachments function deletes any file stored in a table that does not have a corresponding reference stored in ATTACHSYSFILENAME.

See Also

Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," Copy File Attachments

Naming URLs and File Attachments

If you change a URL on the URL Maintenance page from FTP to record, you may have to reboot your application server before you can use the URL in your PeopleCode program.

Converting User Filenames

If a user attaches a file through the browser dialog box generated by the AddAttachment function, the filename ay be converted before that filename is returned to the AddAttachment call for storage.

For example, the file My Resume.doc is returned through the AddAttachment parameter as My_Resume.doc, with the space changed to an underscore.

No comparable mapping occurs for user filenames with the PutAttachment function, which requires the user filename as an input parameter, rather than an output parameter that is returned to the user.

Instead, before storing the ATTACHUSERFILE value, code similar to this should be invoked:

```
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, " ", "_");
 &ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, ";", "_");
 &ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "+", "_");
 &ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "%", "_");
 &ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "&", "_");
 &ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "'", "_");
 &ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "!", "_");
 &ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "@", "_");
 &ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "#", "_");
 &ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "$", "_");
```

CHAPTER 11

Understanding Mobile Pages and Classes

This chapter discusses:

- Mobile component transaction model.
- Mobile device events.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Mobile Agent

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Mobile Classes"

Mobile Component Transaction Model

Mobile pages process transactions similarly to PeopleSoft Pure Internet Architecture.

Unit of Work

A *unit of work* for a mobile application is an instance of a data in a component interface.

Suppose your component interface had the following structure:

```
CUSTOMER
...CUST_ID
   CUST_STATUS
   . . .
   SITE
      SITE_ID
      ADDRESS
      . . .
```

CUSTOMER is the top level of information, the top of the component interface. Your data may include more than one customer, and each customer may have one or more sites.

In a mobile application, you would have two list views based on the example structure, as well as two detail views.

Suppose you have two instances of data, Customer1 and Customer2. A user can change detail about Customer1 and details about Customer1's sites. When the user tries to navigate to Customer2, he or she is prompted to save changes, because navigating to another customer involves leaving one unit of work to begin another unit of work.

The unit of work is finished when the user leaves an instance of the component interface, by either going to another component interface (maybe a peer reference), by going to another instance of data (level zero) of the component interface, or by using the menu navigation to go to another mobile application.

When the user clicks the Save button, the current unit of work is finished and a new unit of work begins.

If an application transfers a user to another object in the same component using TransferMobilePage, the same unit of work is still in existence. If the user is transferred to a different component, a new unit of work is started.

Working Set

A *working set* refers to the objects and data in memory that are available you. It's similar to the component buffer.

When a user selects a mobile application, a list view is opened, containing all the instances of that component interface (all the rows of level zero data). When a user selects one of those instances, that object and all its data becomes part of the working set. The other instances of the component interface are not part of the working set.

In the initial object, only the equivalent of the level zero and level one data is in the working set. However, all the data for the component interface is available to you using PeopleCode. When data is accessed, either through PeopleCode or by the user drilling down to it, it is brought into the working set. For example, if a developer accesses data at level two before a user drills down using a detail view, that data is brought into the working set.

When a user leaves an instance of a component interface, the working set is flushed of all data. A new working set is generated when a user selects a different instance of the component interface.

For example, suppose the top level in the component interface is CUSTOMER, and there are two instances, Customer1 and Customer2. When a user navigates from Customer1 to Customer2, the current working set data is flushed.

Data for a collection is automatically pulled into the working set. When this happens, the OnInit mobile PeopleCode event runs for the data that is brought in. However, data for top-level (level zero) and prompt list views does not initiate the OnInit event. In addition, default values are not set for data displayed for top-level and prompt list views.

Data Saves

Data entered into a mobile application can only be saved either when a user clicks the Save button, or when theDoSave function is used.

You can instantiate, and make changes to, a new component interface using PeopleCode. However, when a user clicks the Save button, those changes are not automatically saved, because a developer-instantiated component interface is not part of the current component.

To save changes made to a component interface instantiated solely through PeopleCode, both of the following must happen:

- You explicitly save the component interface using the Save method or the DoSave function.
- The user must not cancel out of the current component, but must save changes by clicking the Save button.

If the user cancels out of the component, even if you have issued a Save, the data is not saved.

The user is not prompted to save data when leaving the current unit of work if:

- You changed a developer-instantiated component interface and saved those changes with the Save method.
- The user did not make any changes to data in the current component interface.

To ensure that the data actually gets committed to the database, you can force a prompt to occur. This can be done using the following code:

```
%ThisMobileObject.SetModified(True);
```

If a mobile property is based upon a derived field, changing the value of the property does not change the state of the working set. If a user changes a property based upon a derived field, he or she is not prompted to save the mobile page.

Note. If you edit the properties of a component interface reference or data collection, you can mark it as derived. However, the derived flag is ignored for all references (reference properties and collections).

Peer References

A *peer reference* is a related component interface defined on the underlying synchronizable component interface. Peer references can be used in mobile applications as a kind of prompt table for filtering and bringing in additional information into the originating component interface.

Any peer references are automatically brought in as part of the working set. However, only the single instance of the data that relates to the originating component interface is brought in. If you need to work with additional instances of a peer reference, they must be brought into the working set using PeopleCode.

To save data in a peer reference, use the mobile Save method. However, the data for the peer reference is not saved until a user clicks the Save button.

If a change is made to a peer reference by PeopleCode, and no data in the originating component interface has been changed, the user is not prompted to save data.

Field Processing

Mobile pages run in interactive mode. There is no deferred mode processing.

In general, when a user clicks the Save button, fields are verified and processed in the order they appear on the mobile page (tabbing order). However, if a field has PeopleCode associated with it, that field is processed last.

If more than one field has PeopleCode associated with it, and more than one of those fields has been changed by a user, when the user clicks the Save button, these fields are processed after all the fields that do not have PeopleCode associated with them, in the order the user accessed them.

For example, suppose a detail view has an edit box and a button. The user makes a change in the edit box and then clicks the button. When the user clicks the Save button, the edit box and the button are processed after all the other fields on the detail view. The edit box is processed before the button.

Only the following standard system edits are performed by mobile pages:

- Data type checking.
- Formatting.
- Required field.
- Prompt table.

Note. When doing formatting and checking data types, the Custom and Raw Binary field formats are not checked.

Translate values are displayed in a drop-down box for a field, so it is not possible for a user to enter a bad value.

Fields with required yes/no values are displayed as a check box, so it is not possible for a user to enter a bad value.

The only standard system edit that is performed when a user clicks the Save button for a page is the required field edit.

The other system edits are performed in these situations:

- PeopleCode is associated with a property (field) and the user has changed that field, and then leaves it.
- A graphical user interface (GUI) event is processed, such as when a user selects a tab or clicks the Go Back link.

Note. Fraction parts of fields of type Decimal that are too long for the target property declaration are rounded.

See Chapter 6, "PeopleCode and the Component Processor," PeopleSoft Pure Internet Architecture Processing Considerations, page 112.

Field Formats

The Custom and Raw Binary field format types are not supported for mobile pages. The supported field format type edits are:

- Uppercase
- Name
- Phone Number North American
- Zip/Postal Code North American
- SSN
- Mixedcase
- Numbers Only
- SIN
- Phone Number International
- Zip/Postal Code International

Cancel Button

When a user clicks the Cancel button, only the changes for that page are canceled. Changes made to other pages in that instance of the component interface are not rolled back. The parent level of the displaying page is displayed. For example, if a user clicks the Cancel button on a level two detail page, the level one detail view is displayed, with the tab that contains the level two collection active. The user is still in the same unit of work.

A user can edit values displayed in a list view generated by a collection. If a user cancels after changing this type of data, only those changes from that page are rolled back.

Mobile Device Events

This section discusses:

- Event sets.
- OnChange event.
- OnChangeEdit event.
- OnInit event.
- OnObjectChange event.
- OnObjectChangeEdit event.

- OnSaveEdit event.
- OnSavePreChange event.
- OnSavePostChange event.

Event Sets

To open these events from a synchronizable component interface in PeopleSoft Application Designer, select the component interface, then right-click and select View PeopleCode from the pop-up menu.

Synchronizable Component Interface Item	Event Set
Component interface (level zero, top parent) and component interface collection (level one, level two, and so on)	OnInit (Laptop and PDA) OnObjectChange (laptop and PDA) OnObjectChangeEdit (laptop and PDA) OnSaveEdit (laptop and PDA) OnSavePreChange (laptop and PDA) OnSavePostChange (laptop and PDA)
Component Interface Property	OnChange (laptop and PDA) OnChangeEdit (laptop and PDA) **Note.** Clicking a button on a mobile page only initiates the OnChange event for that property.

The mobile device events occur on the mobile device, as the device works with the data from the working set.

Note. Other events are listed on the synchronized component interface, such as OnValidate and OnGetProperty. These events occur on the synchronization server (which is a part of the application server). They do not run on the mobile device.

See Also

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

OnChange Event

The OnChange event is similar to the FieldChange event.

You use OnChange PeopleCode to recalculate mobile page field values, change the appearance of page controls, run upon a button being clicked, or perform other processing that results from a field change other than data validation.

To validate the contents of the field, use the OnChangeEdit event. The OnChange event occurs on the specific mobile page field and row that just changed.

Do not use error or warning statements in OnChange PeopleCode: these statements cause a runtime error that forces the user to cancel the page without saving changes.

OnChange PeopleCode is often paired with OnInit PeopleCode. In these OnInit and OnChange pairs, the OnInit PeopleCode checks values in the working set and initializes the state or value of properties (fields) accordingly. OnChange PeopleCode then rechecks the values in the working set during page execution and resets the state or value of properties (fields).

If a user has changed a property and presses TAB to leave the row (or page), the OnObjectChange event initiates after OnChange.

The OnObjectChange event does not run if the user clicks either the Cancel button or the browser's Back button.

OnChangeEdit Event

The OnChangeEdit event is similar to the FieldEdit event.

You use OnChangeEdit PeopleCode to validate the contents of a mobile page field, supplementing the standard system edits. If the data that the user entered for that field does not pass the validation, the PeopleCode program should display a message using the Error statement, which redisplays the page, displays an error message, and turns the mobile page field red.

To permit the edit, but alert the user to a possible problem, use a Warning statement instead of an Error statement. A Warning statement displays a warning dialog box with OK and Explain buttons. It permits mobile page field contents to be changed and continues processing as usual after the user clicks OK.

If the validation is checking for consistency across multiple page fields, use OnSaveEdit PeopleCode instead of OnChangeEdit.

OnInit Event

The OnInit event is similar to the RowInit event.

The OnInit event occurs as each mobile object is loaded into the working set. This also applies when a newly created object is loaded into the working set.

Note. The *mobile object* represents the PeopleSoft Pure Internet Architecture equivalent of a row of data, and can be used at any level in the data hierarchy: level zero, level one, level two, and so on. The mobile object is displayed in the detail view of a mobile page.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "Mobile Classes," Understanding the Mobile Device and the Component Interface.

Since the working set contains only objects that are actually accessed, the OnInit events occur over time. In PeopleSoft Pure Internet Architecture, the RowInit events are called all together, when the component buffers are built.

OnChange PeopleCode is often paired with OnInit PeopleCode.

Note. Do not use Error or Warning statements in OnInit PeopleCode: these cause a runtime error and force the user to cancel the component without saving.

OnObjectChange Event

The OnObjectChange event occurs when a user navigates away from a mobile object. This includes such actions as moving from one tab to another on a mobile page, pressing TAB to leave a row on a detail view, or clicking the Save button. OnObjectChange is only triggered by changes to the GUI.

OnObjectChange only runs for rows that have actually changed. For example, if the user selects a check box on a row in a collection, and the OnChange PeopleCode associated with the check box changes every other row in the collection, OnObjectChange only runs for the row that was changed by the GUI.

If a user changes the main object as well as a row in a collection list view that both have OnObjectChange PeopleCode, and then the user leaves the main object, the OnObjectChange event runs for the row before it runs for the main object.

The OnObjectChange event also runs for the main object if a change has been made, and the user does not leave the main object but clicks the Save button. If the user has made a change to a row in a collection list view, and then clicks the Save button, OnObjectChange also runs, because the user left the row.

If a user changes a property on a collection list view row that has the OnChange event, and then the user leaves the row, OnChange runs before the OnObjectChange event for the row (if there is OnChange PeopleCode associated with the property). If a user changes a property on a collection list view row that has the OnChange event, and then leaves the row by selecting a check box on a second row that has OnChange PeopleCode, OnChange runs for the property on the first row, then the OnObjectChange event runs for the first row, and then OnChange runs on the check box.

If a user changes a property on a collection list view row that has the OnChange event, and then leaves the row, OnChange runs before the OnObjectChange event for the row, if OnObjectChange is present. If a user changes a property on a collection list view row that has the OnChange event, and then leaves the row by selecting a check box on a second row that has OnChange, OnChange runs for the property on the first row, then the OnObjectChange event runs for the first row, and finally OnChange runs on the check box.

The OnObjectChange event does not run if a user clicks either the Cancel button or the browser's Back button.

OnObjectChangeEdit Event

The OnObjectChangeEdit event occurs when a user navigates away from a mobile object. This includes such actions as moving from one tab to another on a mobile page,pressing TAB to leave a row on a detail view, or clicking the Save button. OnObjectChangeEdit is only triggered by changes to the GUI.

The OnObjectChangeEdit event runs before the OnObjectChange event.

The OnObjectChangeEdit event only runs for rows that have actually changed. For example, if a user selects a check box on a row in a collection, and the OnChange PeopleCode associated with the check box changes every other row in the collection, OnObjectChangeEdit only runs for the row that was changed by the GUI.

If a user changes the main object as well as a row in a collection list view that both have OnObjectChangeEdit PeopleCode, and then the user leaves the main object, the OnObjectChangeEdit event runs for the row before it runs for the main object.

The OnObjectChangeEdit event also runs for the main object if a change has been made, and the user does not leave the main object but clicks the Save button. If the user has made a change to a row in a collection list view, and then clicks the Save button, the OnObjectChangeEdit event also runs, because the user left the row.

If a user changes a property on a collection list view row that has the OnChange event, and then the user leaves the row, OnChange runs before the OnObjectChangeEdit event for the row (if there is OnChange PeopleCode associated with the property). If a user changes a property on a collection list view row that has the OnChange event, then leaves the row by selecting a check box on a second row that has OnChange PeopleCode, OnChange runs for the property on the first row, then the OnObjectChangeEdit event runs for the first row, and then OnChange runs on the check box.

If a user changes a property on a collection list view row that has the OnChange event, and then leaves the row, OnChange runs before the OnObjectChangeEdit event for the row, if OnObjectChangeEdit is present. If a user changes a property on a collection list view row that has the OnChange event, and then leaves the row by selecting a check box on a second row that has OnChange, OnChange runs for the property on the first row, then the OnObjectChangeEdit event runs for the first row, and finally OnChange runs on the check box.

The OnObjectChangeEdit event does not run if the user clicks either the Cancel button or the browser's Back button.

OnSaveEdit Event

The OnSaveEdit event is similar to the SaveEdit event. The OnSaveEdit event runs for every modified object in the working set.

The OnSaveEdit event occurs whenever the user tries to save the component (clicks the Save button). Use OnSaveEdit PeopleCode to validate the consistency of data in the mobile object user-defined properties. Whenever a validation involves more than one property, use OnSaveEdit PeopleCode. If a validation involves only one property, use OnChangeEdit PeopleCode.

An Error statement in OnSaveEdit PeopleCode displays a message and redisplays the mobile page without saving data. A warning enables the user to click OK and save the data, or click Cancel and return to the mobile page without saving.

OnSavePostChange Event

The OnSavePostChange event is similar to the SavePostChange event.

The OnSavePostChange event occurs after the object is saved. Use OnSavePostChange PeopleCode to update peer references.

Note. An error or warning in OnSavePostChange PeopleCode causes a runtime error, forcing the user to cancel the mobile page without saving changes. Avoid errors and warnings in this event.

The system issues a Structured Query Language (SQL) Commit statement after OnSavePostChange PeopleCode completes successfully.

OnSavePreChange Event

The OnSavePreChange event is similar to the SavePreChange event.

The OnSavePreChange event runs after tje OnSaveEdit event completes without errors. OnSavePreChange PeopleCode provides one final opportunity to manipulate data before the system updates the database on the mobile device.

Note. An error or warning in OnSavePostChange PeopleCode causes a runtime error, forcing the user to cancel the mobile page without saving changes. Avoid errors and warnings in this event.

Mobile Event Flow

This section discusses:

- General event order.
- Event order when a page is saved.

General Event Order

Mobile events run only if there is a PeopleCode program associated with the event. The following is the general running order of the mobile events.

1. Initialize the mobile object data with the OnInit event.

   This event occurs when a mobile object is first entered into the working set, either from the database or when it is created.

2. Change property values with the OnChangeEdit or OnChange events.

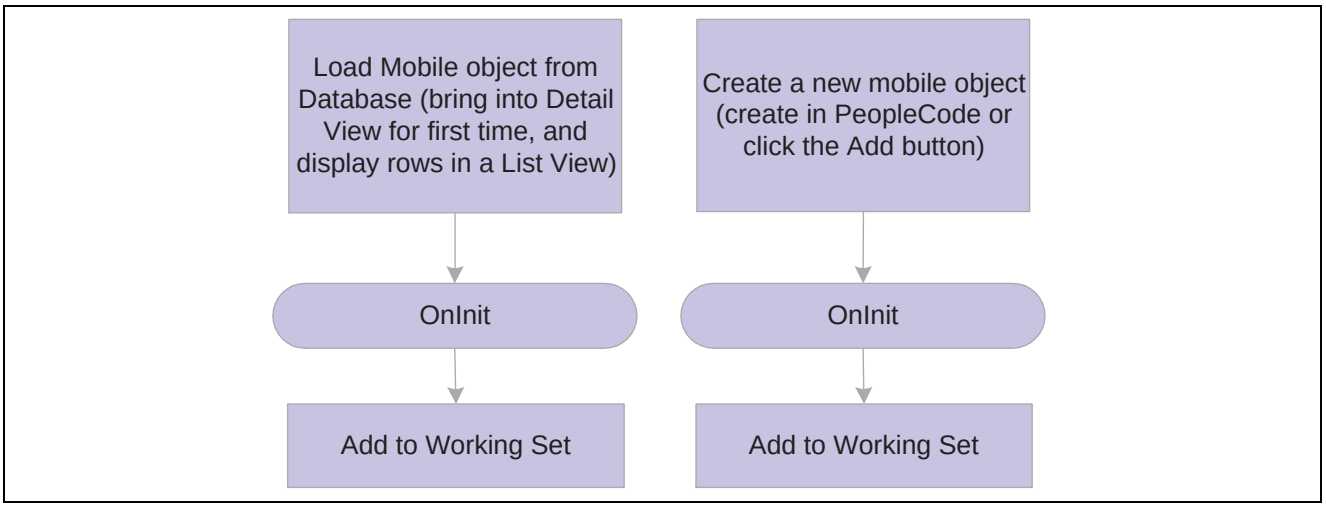
   These events run only in response to a change in the GUI.

3. Navigate off a mobile object that has had its values changed with the OnObjectChangeEdit or OnObjectChange events.

   These events run only in response to a change in the GUI.

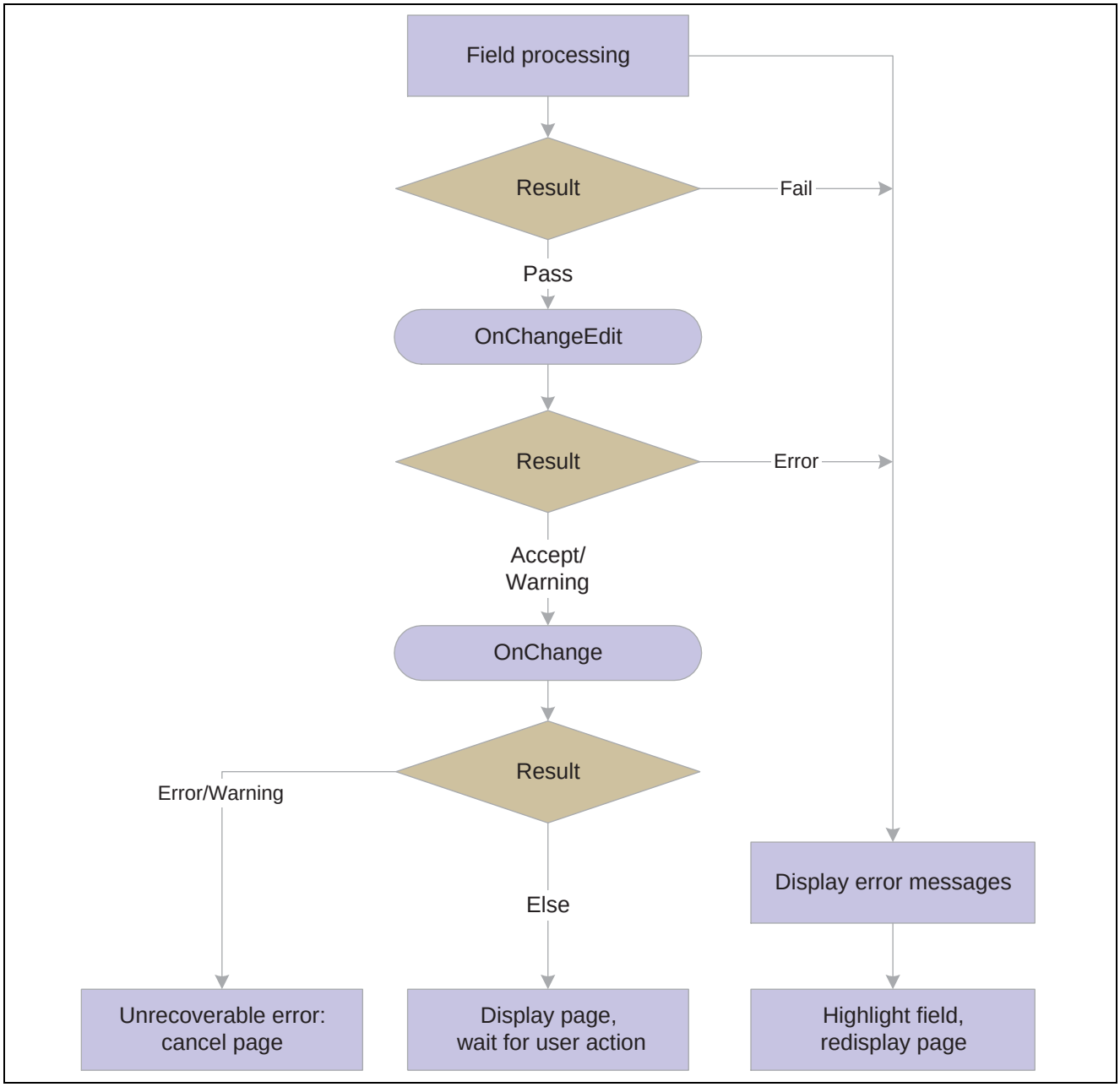
4. Save the page with the OnSaveEdit, OnSavePreChange, and OnSavePostChange events.

The following diagram shows how data is initialized and brought into the working set:



Reading a mobile object into the working set

The following diagram shows the event flow for when a property changes:



Changing a mobile page field (component interface property)

Event Order When a Page Is Saved

When a user clicks the Save button, the mobile save events operate as follows:

1. The OnChangeEdit, OnChange, OnObjectChangeEdit and OnObjectChange events run on the current mobile objects as applicable.
2. The system obtains the top parent on the current mobile object that is displayed in the mobile page detail view.
3. Working from the top parent down, a depth-first, top-to-bottom traversal of the working set hierarchy occurs, and for every modified mobile object found, the system checks the required fields.

   If a required field is not filled in correctly, the save is stopped and the field is highlighted in red.

4. Working from the top parent down, a depth-first, top-to-bottom traversal of the working set hierarchy occurs.

   For every modified mobile object found, the OnSaveEdit PeopleCode runs.

5. Working from the top parent down, a depth-first, top-to-bottom traversal of the working set hierarchy occurs.

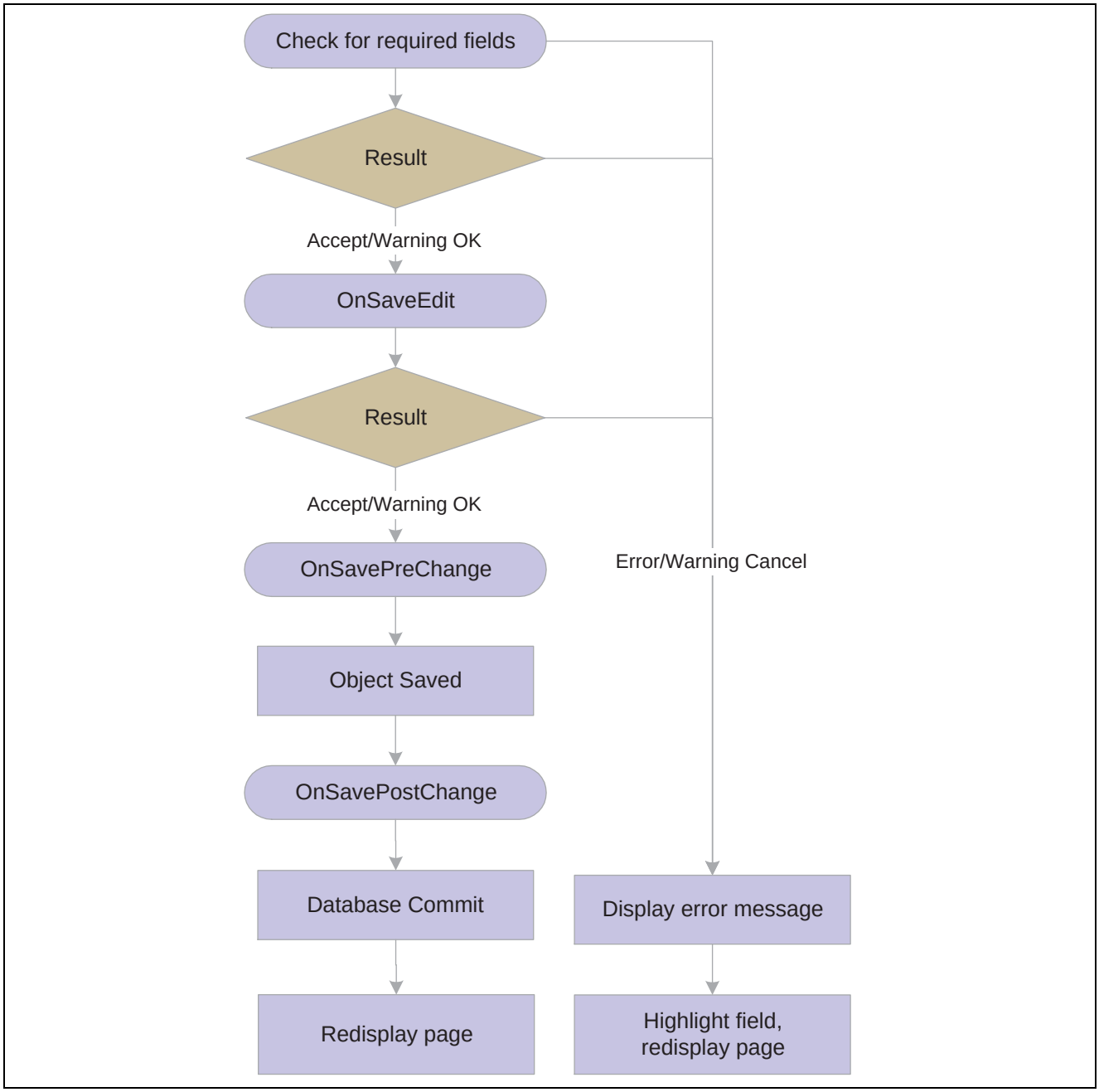
   For every modified mobile object found, the OnSavePreChange PeopleCode runs.

6. Working from the top parent down, a depth-first, top-to-bottom traversal of the working set hierarchy occurs.

   For every modified mobile object found, the OnSavePostChange event PeopleCode runs.

The depth-first, top-to-bottom traversal of the working set hierarchy works by processing each child mobile object in the working set. For each child, processing is recursively invoked, causing that child to be processed. Then, for each of the child's children, processing is also recursively invoked.

The following is a diagram of the save process:



Saving a mobile object

CHAPTER 12

Accessing PeopleCode and Events

This chapter provides an overview of PeopleCode, events and how PeopleCode programs are associated with events, as well as discusses how to:

- Understand automatic backup of PeopleCode.
- Access PeopleCode in PeopleSoft Application Designer.
- Access record field PeopleCode.
- Access component record field PeopleCode.
- Access component record PeopleCode.
- Access component PeopleCode.
- Access page PeopleCode.
- Access menu item PeopleCode.
- Access message PeopleCode.
- Copy PeopleCode with a parent definition.
- Upgrade PeopleCode programs.

Understanding PeopleCode Programs and Events

Every PeopleCode program is associated with some aspect of a PeopleSoft Application Designer definition and an event. Events are predefined points either in the Component Processor flow or in the program flow. As each event is encountered, it fires on each component, triggering any PeopleCode program associated with that component and that event. Each definition in PeopleSoft Application Designer can have an *event set*, that is, a group of events appropriate to that definition. A definition can have zero or one PeopleCode programs for each event in its event set.

Some definitions have events that fall outside the Component Processor flow. These definitions include Application Engine programs, component interfaces, messages, and application packages. In addition, security has a signon “event”. These events are described in the documentation for the definition or topic.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Component Interface Classes"

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Application Classes"

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker, "Defining Message Channels and Messages," Defining Message Definitions

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Engine, "Creating Application Engine Programs," Specifying PeopleCode Actions

Enterprise PeopleTools 8.45 PeopleBook: Security Administration, "Understanding PeopleSoft Security"

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Mobile Agent, "Understanding PeopleSoft Mobile Agent"

Understanding Automatic Backup of PeopleCode

A PeopleCode program is automatically saved to a file while you're working on it. This checkpoint occurs at the following times:

- Every 10 keystrokes.
- On a save command, just prior to the save being executed (in case the save does not actually execute because the code is invalid).
- When another PeopleCode program is selected to be edited (if you have two PeopleCode editor windows open at the same time, and you move from one to the other).

The file is saved to your temp directory (as specified in your environment), in a file with the following name:

```
PPCMMDDYY_HHMMSS.txt
```

where MMDDYY represents the month, date, and year, respectively, of the checkpoint, and HHMMSS represents the hour, minute, and second, respectively.

The top of the checkpoint file contains the following information:

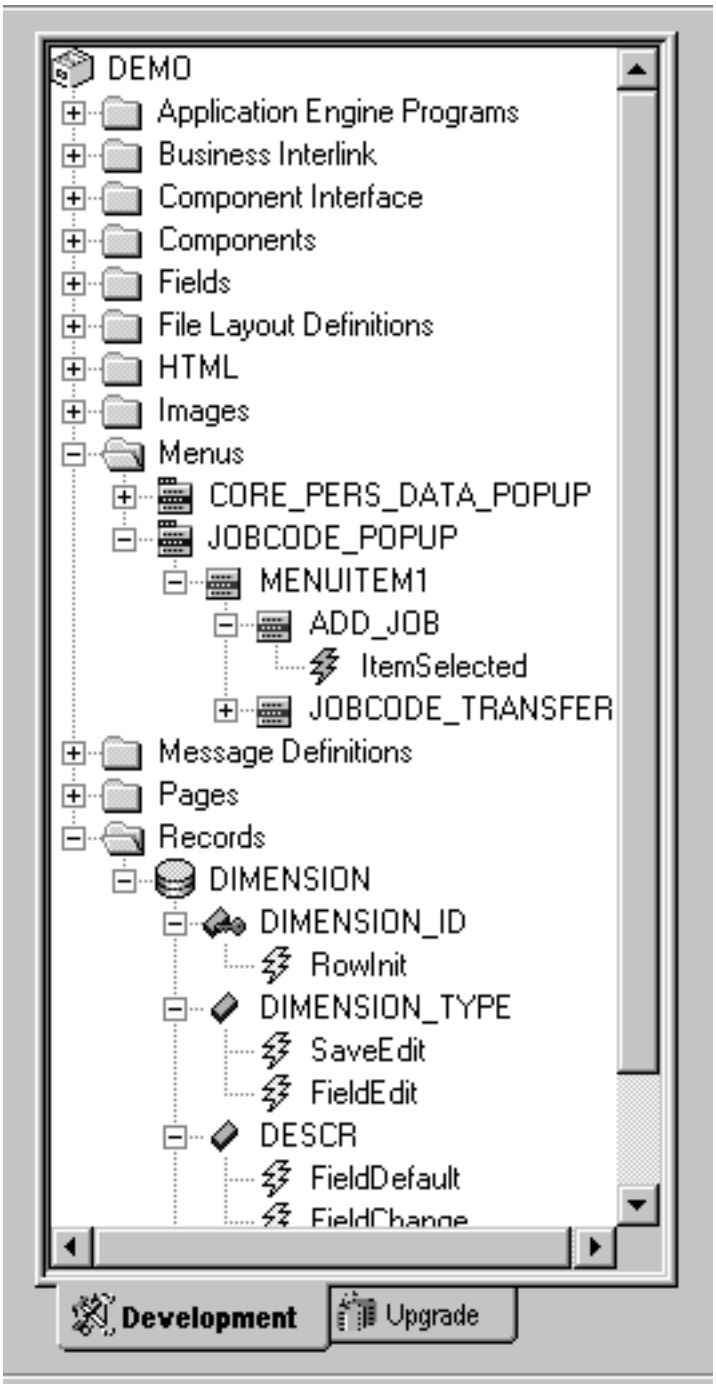
```
[PeopleCode Checkpoint File]
[RECORD.recordnameFIELD.fieldnameMETHOD.eventname]
```

If your PeopleCode program is saved successfully, any checkpoint files associated with that program are automatically deleted.

Accessing PeopleCode in PeopleSoft Application Designer

You can access PeopleCode associated with PeopleSoft Application Designer definitions in several ways.

With record fields and pop-up menu items, the Project view displays PeopleCode programs within the project hierarchy using a lightning bolt symbol. The programs are children of the fields and pop-up menu items with which they're associated, and are named according to their associated events, such as ItemSelected, RowInit, or SaveEdit, as shown in the following illustration. Double-click a record field or pop-up menu item program in the Project view to start the PeopleCode Editor and load that program for editing.



PeopleCode programs in the Project view hierarchy

You can associate PeopleCode with other types of definitions, such as:

- Components
- Pages
- Component interfaces
- Messages

Such PeopleCode programs do not appear in the Project view. Instead, right-click the definition's name and select View PeopleCode. You can also access these programs from their associated definitions.

PeopleCode can also be associated with:

- Component records (specific records included in components).
- Component record fields (specific record fields included in components).

Because component record fields and component records do not appear in the Project view, you must access their associated programs through their parent definitions.

See Also

Chapter 12, "Accessing PeopleCode and Events," Accessing Record Field PeopleCode, page 208

Chapter 12, "Accessing PeopleCode and Events," Accessing Component PeopleCode, page 213

Accessing Record Field PeopleCode

This section provides an overview of the record field event set and discusses how to:

- Access record field PeopleCode from a record definition.
- Access record field PeopleCode from a page definition.

Understanding Record Field PeopleCode

A record is a table-level definition. There are different types of record definitions, such as Structured Query Language (SQL) table, dynamic view, derived/work, and so on.

Record fields are child definitions of records. Record field PeopleCode programs are child definitions of record fields. A record field can have zero or one PeopleCode programs for each event in the record field event set.

The following are the events that are associated with a record field:

- FieldChange Event
- FieldDefault Event
- FieldEdit Event
- FieldFormula Event
- RowInit Event
- RowSelect Event
- RowDelete Event
- PrePopup Event
- SaveEdit Event
- SavePreChange Event
- Workflow Event
- SavePostChange Event
- SearchInit Event

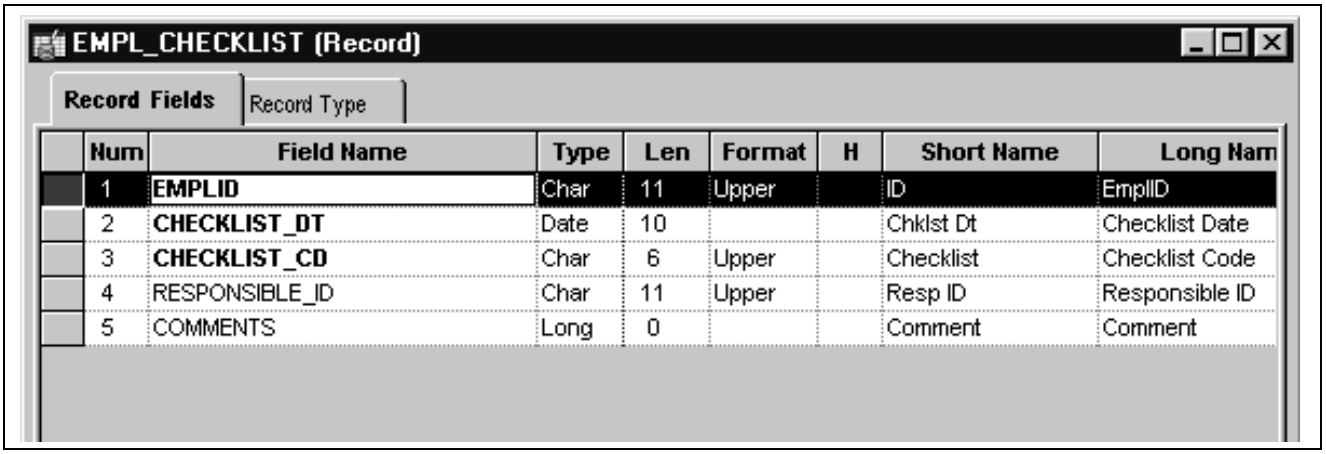
- SearchSave event

See Also

Chapter 6, “PeopleCode and the Component Processor,” page 81

Accessing Record Field PeopleCode from a Record Definition

Record definition fields that have PeopleCode associated with them appear bold in record views.



Num	Field Name	Type	Len	Format	H	Short Name	Long Nam
1	**EMPLID**	Char	11	Upper		ID	EmplID
2	**CHECKLIST_DT**	Date	10			Chklst Dt	Checklist Date
3	**CHECKLIST_CD**	Char	6	Upper		Checklist	Checklist Code
4	RESPONSIBLE_ID	Char	11	Upper		Resp ID	Responsible ID
5	COMMENTS	Long	0			Comment	Comment

Record definition with three fields with PeopleCode

In the previous example, the first three fields (in bold) have PeopleCode associated with them. If you expand the subrecords in a record definition, any fields in the subrecord that have PeopleCode associated with them appear in bold.

To access record field PeopleCode from an open record definition:

1. Click the PeopleCode Display button on the toolbar.

   A grid appears with a column for each event in the record field event set. Each cell represents a field-event combination. The column names are abbreviations of the record field event names, for example, *FCh* for the FieldChange event and *RIn* for the RowInit event. A check mark appears in the appropriate cell for each field/event combination that has an associated PeopleCode program.

2. Access the PeopleCode.

   You can access the PeopleCode for a cell using one of these methods:

   - Double-click the cell.
   - Right-click the cell and select View PeopleCode.
   - Select View, PeopleCode.

The PeopleCode Editor appears. If the field/event combination has an associated program, it appears in the editor.

See Also

Chapter 12, “Accessing PeopleCode and Events,” Understanding Record Field PeopleCode, page 208

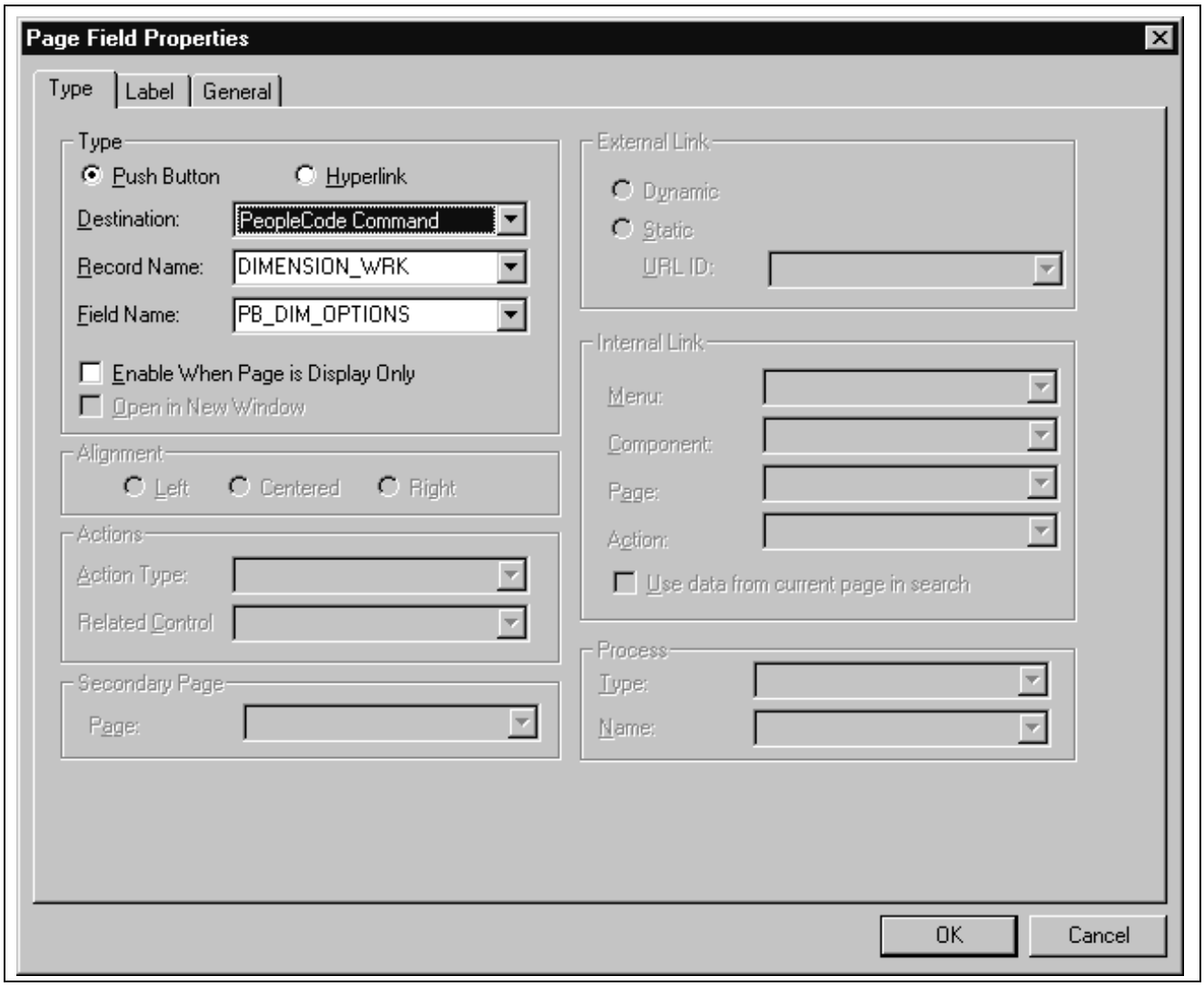
Chapter 13, “Using the PeopleCode Editor,” page 219

Accessing Record Field PeopleCode from a Page Definition

You can associate a PeopleCode program with any page control that you can associate with a record field.

To access record field PeopleCode from a page definition, right-click a page control and select View Record PeopleCode. The PeopleCode Editor appears, displaying the first event in the event set associated with that control's underlying record field.

Button controls are a special case. You can associate a PeopleCode program with a button only if its destination is defined as PeopleCode Command. When the user clicks a button defined using this method, the FieldEdit and FieldChange events are triggered, so the PeopleCode must be associated with one of those two events. Typically, you use the FieldChange event. The following example shows button properties:



Page Field Properties dialog box for buttons

To define a command button:

1. In the page definition, double-click the button to access its properties.
2. Select *PeopleCode Command* as the button destination.
3. Select the record and field with which your button and PeopleCode are associated.

It's best to associate the button with a derived or work record field, which separates its PeopleCode from the PeopleCode associated with any of the page's other underlying record fields. You can then store generic PeopleCode with this field so you can reuse it with button on other pages.

4. Click OK to return to the page.

   Right-click the button and select View PeopleCode to access the PeopleCode Editor.

See Also

Chapter 13, "Using the PeopleCode Editor," page 219

Accessing Component Record Field PeopleCode

This section provides an overview of component record field PeopleCode and discusses how to access component record field PeopleCode.

Understanding Component Record Field PeopleCode

Component record field PeopleCode is associated with a record field, but only with respect to a component and one of its events. Use this type of association to tailor your programs to a particular component. This PeopleCode is only accessible through the component structure view, not from a record definition.

The following events are associated with a component record field:

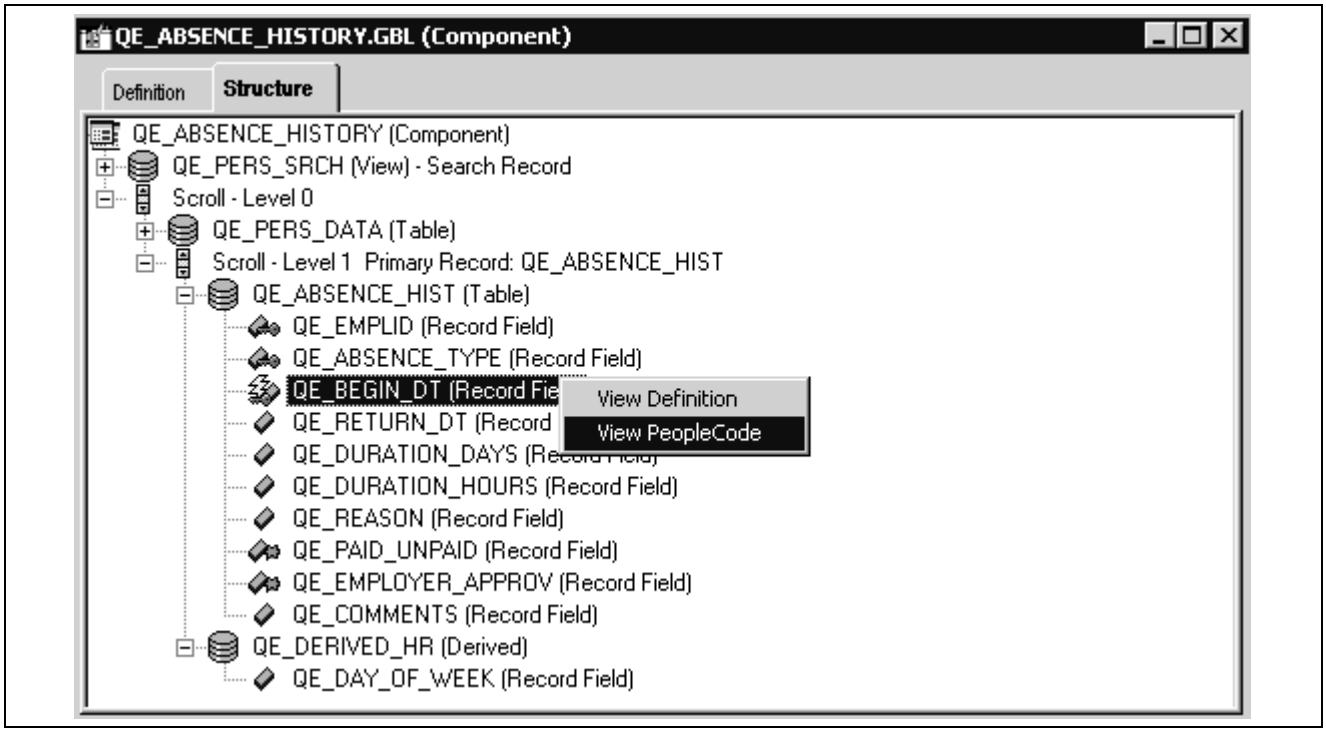
- FieldChange Event
- FieldDefault Event
- FieldEdit Event
- PrePopup Event

See Also

Chapter 6, "PeopleCode and the Component Processor," page 81

Accessing Component Record Field PeopleCode

To access PeopleCode associated with a component record field, open the component, click the Select tab, select a field, right-click the field name, and select View PeopleCode. A lightning bolt appears next to the field name if PeopleCode is associated with the field at the component level. If PeopleCode is associated with the field at the record level, a lightning bolt does not appear, as shown in the following illustration:



Accessing component record field PeopleCode from the component structure

Note. The Structure tab only displays the runtime state of the PeopleCode. That is, it only displays record field PeopleCode. For example, PeopleCode programs that are orphaned as a result of a page definition change do not appear in the Structure tab. On the other hand, these orphaned PeopleCode programs do appear in the PeopleCode Editor, which displays the design-time view of PeopleCode.

The PeopleCode Editor appears. If that field has associated PeopleCode, the first program in the component record field event set appears in the editor.

See Also

Chapter 13, “Using the PeopleCode Editor,” page 219

Chapter 12, “Accessing PeopleCode and Events,” Accessing Record Field PeopleCode, page 208

Accessing Component Record PeopleCode

This section provides an overview of component record PeopleCode and discusses how to access component record PeopleCode.

Understanding Component Record PeopleCode

Component record PeopleCode is associated with a record definition, but only with respect to a component and one of its events. Use this type of association to tailor programs to a particular component. This PeopleCode is directly accessible through the component structure view, not from the record definition.

Search records and non-search records in components have different associated event sets. The following events are associated with component search records:

- SearchInit Event
- SearchSave Event

The following events are associated with component non-search records:

- RowDelete Event
- RowInit Event

  In rare circumstances, the Component Processor does not run RowInit PeopleCode for some record fields. The Component Processor runs RowInit PeopleCode when it loads the record from the database. However, in some cases, the record can be initialized entirely from the keys for the component. When this happens, RowInit PeopleCode is not run.
- RowSelect Event
- SaveEdit Event
- SavePostChange Event
- SavePreChange Event

See Also

Chapter 6, "PeopleCode and the Component Processor," page 81

Accessing Component Record PeopleCode

To access PeopleCode associated with a component record, open the component's structure view, select a record, right-click the record name, and select View PeopleCode.

The PeopleCode Editor appears. If the record has associated PeopleCode, the first program in the component record event set appears in the editor.

See Also

Chapter 13, "Using the PeopleCode Editor," page 219

Accessing Component PeopleCode

This section provides an overview of component PeopleCode and discusses how to access component PeopleCode.

Understanding Component PeopleCode

Component PeopleCode is associated with a component definition and an event.

The following events can be associated with a component:

- PostBuild Event
- PreBuild Event
- SavePostChange Event
- SavePreChange Event

- Workflow Event

See Also

Chapter 6, “PeopleCode and the Component Processor,” page 81

Accessing Component PeopleCode

To access PeopleCode associated with a component, open its structure view, select the component name, right-click the name, and select View PeopleCode.

The PeopleCode Editor appears. If the component has associated PeopleCode, the first program in the component event set appears in the editor.

See Also

Chapter 13, “Using the PeopleCode Editor,” page 219

Accessing Page PeopleCode

This section provides an overview of page PeopleCode and discusses how to access page PeopleCode.

Understanding Page PeopleCode

Page PeopleCode is associated with a page definition. The page event set consists of a single event, the Activate event, which fires every time the page is activated. This event is valid only for pages that are defined as standard or secondary, and is not supported for subpages.

See Also

Chapter 6, “PeopleCode and the Component Processor,” page 81

Accessing Page PeopleCode

To access PeopleCode associated with a page, right-click any part of the page’s definition and select View Page PeopleCode.

Note. Page PeopleCode can only be accessed in this manner. You cannot access Page PeopleCode from the component definition Structure tab, from a project, and so on.

The PeopleCode Editor appears. If the page has associated PeopleCode, it appears in the editor.

Note. The term *page PeopleCode* refers to PeopleCode programs owned by pages. Do not to confuse page PeopleCode with PeopleCode properties related to the appearance of pages, such as the Visible Page Class property.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, “Page Class”

Chapter 13, “Using the PeopleCode Editor,” page 219

Accessing Menu Item PeopleCode

This section provides an overview of menu item PeopleCode and discusses how to:

- Define PeopleCode pop-up menu items.
- Access menu item PeopleCode.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer, "Creating Menu Definitions"

Chapter 13, "Using the PeopleCode Editor," page 219

Understanding Menu Item PeopleCode

PeopleTools menus are either one of two types, pop-up and standard, both of which are standalone definitions in the project hierarchy. However, you can only associate PeopleCode with menu items in pop-up menus.

The menu item event set consists of a single event, the ItemSelected Event. This event fires whenever an user selects a menu item from a pop-up menu.

Note. Do not confuse menu item PeopleCode with PeopleCode functions related to the appearance of menu items, such as CheckMenuItem.

See Also

Chapter 6, "PeopleCode and the Component Processor," ItemSelected Event, page 118

Defining PeopleCode Pop-Up Menu Items

To define a PeopleCode pop-up menu item:

1. In the open pop-up menu definition, double-click the menu item to access its properties.

   If you're creating a new menu item, double-click the empty rectangle at the bottom of the pop-up menu.
2. The Menu Item Properties dialog box appears.

   If this is a new menu item, enter a name and a label for the item.
3. Select *PeopleCode* from the Type group.
4. Click OK to close the Menu Item Properties dialog box.

Accessing Menu Item PeopleCode

To access pop-up menu item PeopleCode:

1. Open the pop-up menu definition.
2. Right-click the menu item and select View PeopleCode.

   The PeopleCode Editor appears with that menu item's associated program, if any, displayed.

Accessing Message PeopleCode

This section provides an overview of message PeopleCode and discusses how to access message PeopleCode.

Understanding Message PeopleCode

These events are not considered part of the Component Processor flow, so they're documented separately from the majority of PeopleCode events. The following events are associated with messages:

- OnPublishTransform Event
- OnSubscribeTransform Event
- OnRouteSend Event
- OnRouteReceive Event
- OnRequest Event

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker, "Defining Message Channels and Messages"

Accessing Message PeopleCode

To access PeopleCode associated with a message, open the message definition, right-click anywhere in the message structure display, and select View PeopleCode.

The PeopleCode Editor appears. If the message definition has associated programs, the first one in the application message event set appears in the editor.

Each message subscription entry in a message definition represents a PeopleCode program. To access that program, open the message definition, select the subscription, right-click the subscription name, and select View PeopleCode.

The PeopleCode Editor appears, with the message subscription's associated program displayed in the editor.

See Also

Chapter 13, "Using the PeopleCode Editor," page 219

Copying PeopleCode with a Parent Definition

When you copy a PeopleSoft Application Designer definition that contains PeopleCode, you can choose whether to copy all PeopleCode programs and the definition. Each copy of the definition receives a separate copy of the PeopleCode programs.

To copy a definition with its PeopleCode:

1. Open the definition you want to copy.
2. Select File, Save As.

   The Save As dialog appears. Type a name for the new definition in the dialog box.

3. Click OK, then click Yes to copy the PeopleCode.

   Click Yes to copy all PeopleCode associated with the definition.

Upgrading PeopleCode Programs

You can upgrade PeopleCode programs independently of the definitions with which they're associated. Refer to the upgrade instructions for your product for more details.

CHAPTER 13

Using the PeopleCode Editor

This chapter discusses how to:

- Navigate between PeopleCode programs.
- Use the PeopleCode Editor.
- Generate PeopleCode using drag-and-drop.

Navigating Between PeopleCode Programs

After you access a PeopleCode program associated with a PeopleSoft Application Designer definition, you can access programs associated with other related definitions without having to close the editor window.

This section provides an overview of the PeopleCode Editor window and discusses how to:

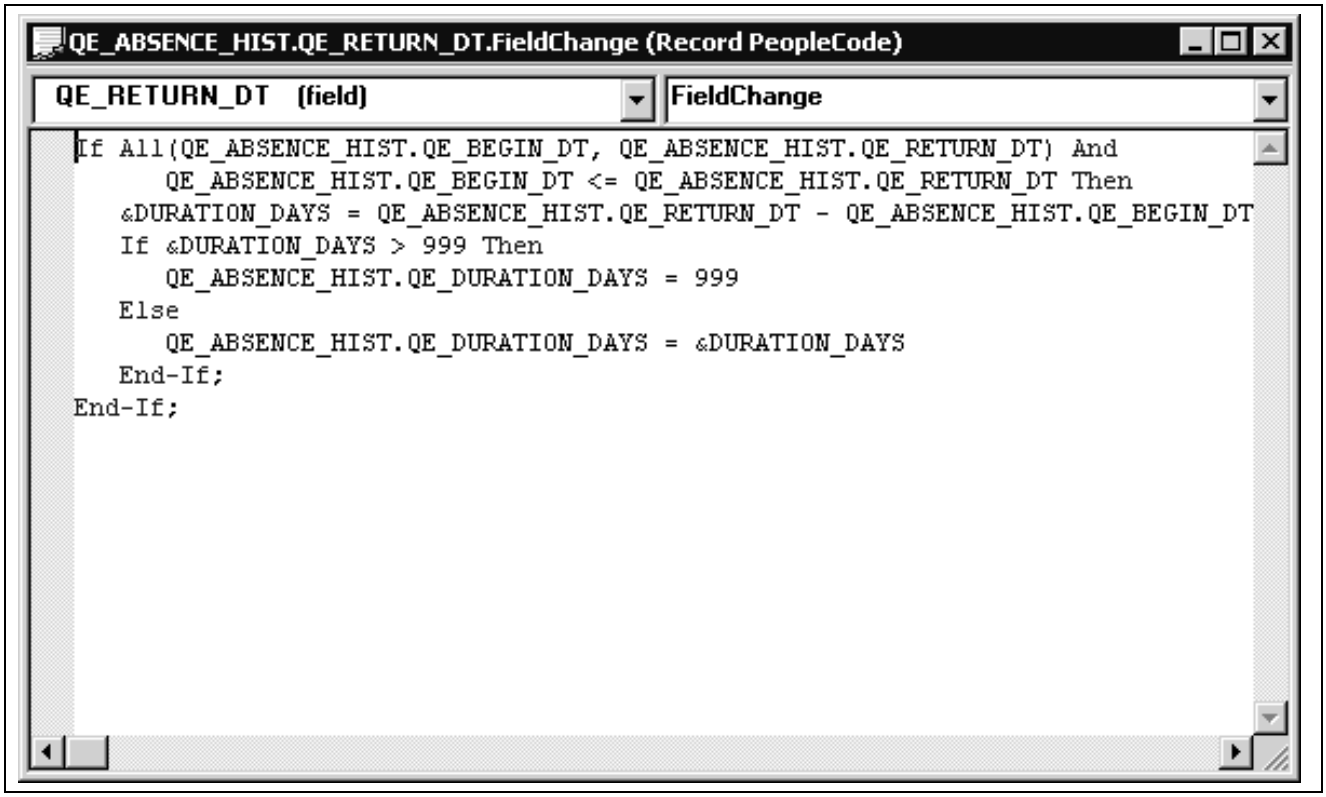
- Navigate between programs associated with a definition and its children.
- Navigate between programs associated with a definition's event set.

See Also

Chapter 12, "Accessing PeopleCode and Events," page 205

Understanding the PeopleCode Editor Window

PeopleSoft Application Designer supplies an independent editor window for each parent definition, such as a record, definition, menu, or message, for which you invoke the editor. The editor window's title bar displays the name and type of the parent definition, as shown in the following illustration:



PeopleCode Editor window with record field PeopleCode

The editor window contains the main edit pane, the drop-down definition list at the upper-left, and the drop-down event list at the upper-right. The drop-down lists enable you to navigate directly to the PeopleCode associated with related child definitions, for example, fields within a record and their event sets.

Note. When you make a selection from either drop-down list box, your selected entry has a yellow background, indicating that you must click the edit pane before you can start typing.

You can open as many editor windows as you want and resize them in PeopleSoft Application Designer. Each line of code wraps automatically based on the window's current width. A vertical scroll bar appears if the program has more lines than the editor can display in the edit pane.

Note. You cannot open two editor windows for a single parent definition, or for any two of its child definitions.

See Also

Chapter 13, "Using the PeopleCode Editor," Navigating Between Programs Associated With a Definition and Its Children, page 220

Chapter 13, "Using the PeopleCode Editor," Navigating Between Programs Associated With Events, page 221

Navigating Between Programs Associated With a Definition and Its Children

You use the drop-down definition list to navigate between PeopleCode programs that are associated with a parent definition and its children. The list displays the complete hierarchy of child definitions to which you can navigate; bold items have PeopleCode associated with at least one event in the item's event set. The structure of the definition list depends on the type of parent definition. Parent definitions include:

- Records.

  Select record fields from the record drop-down list. The record name appears at the top of the list as a visual clue to clarify the location of the record fields, but you cannot associate PeopleCode with a record.

- Components.

  Select component records and component record fields from the component drop-down list.

- Pages.

  Select the page definition from the page drop-down list.

- Pop-up menus.

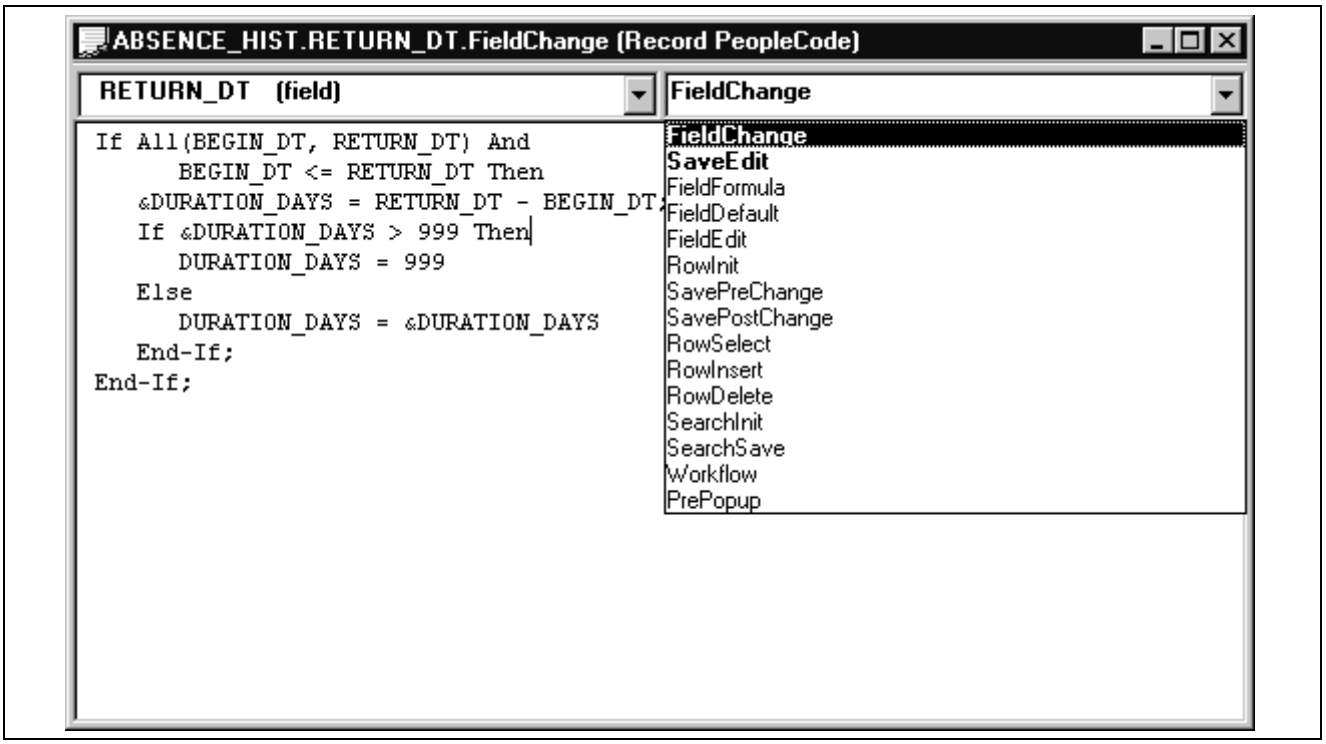
  Select pop-up menu items from the menu drop-down list. The menu and menu bars appear in the list as visual clues, but you cannot associate PeopleCode with these elements.

- Messages.

  Select messages or message subscriptions from the message PeopleCode drop-down definition list. The message subscriptions *are* PeopleCode programs, so they are always bold.

Navigating Between Programs Associated With Events

Use the PeopleCode Editor's drop-down event list to select an event from the event set of the currently selected definition. Use this event list to navigate between PeopleCode programs that are associated with that definition. For every definition-event combination with associated PeopleCode, the event name is displayed in bold, and it appears at the top of the event list, as shown in the following illustration:



Selecting an event from the PeopleCode Editor

See Also

Chapter 12, “Accessing PeopleCode and Events,” page 205

Chapter 6, “PeopleCode and the Component Processor,” page 81

Using the PeopleCode Editor

This section provides an overview of the PeopleCode Editor and color-coded language elements and discusses how to:

- Write and edit PeopleCode.
- Format code automatically.
- Use drag-and-drop editing.
- Access PeopleCode external functions.
- Access definitions and associated PeopleCode.
- Access help.
- Set up help.
- Change colors in the PeopleCode Editor.
- Select a font for the PeopleCode Editor.
- Change word wrap in the PeopleCode Editor.
- Use PeopleCode Event properties.

Understanding the PeopleCode Editor

The PeopleCode Editor works much like any other text editor, but has capabilities specifically geared toward the PeopleTools environment. Some of its features include:

- Editing functions are integrated with the menus and toolbar of Application Designer and are also accessible from a pop-up window.
- It checks, formats, and saves all programs associated with PeopleSoft Application Designer definitions simultaneously when any definition is saved.
- It includes a Validate Syntax command for checking and formatting a single PeopleCode program without saving.
- It supports standard Microsoft Windows drag-and-drop editing.
- It supports color-coding for the different elements of the PeopleCode language.
- It supports word wrap based on either the size of the editor window or a specific number of characters per line.
- You can open separate instances of the editor simultaneously, and you can use a drag-and-drop text operation between programs.
- You can open the definition with which the current set of PeopleCode programs is associated from within the PeopleCode Editor.
- You can open a field, record, page, file layout, message definition, or other definitions from a PeopleCode reference to the field, record, page, file layout, or message, and so on.

- You can access PeopleCode programs associated with a field, record, page, file layout message definition, or other definitions from a PeopleCode reference to the field, record, page, file layout, or message, and so on.
- You can open a PeopleCode Editor window containing an external function definition from a function declaration or function call.
- You can press F1 with the cursor in a PeopleCode built-in function, method, meta-SQL, and so on, to open the PeopleSoft help for that item.

Writing and Editing PeopleCode

The PeopleCode Editor supports standard editing function commands such as Save, Cancel, Cut, Copy, Paste, Find, Replace, and Undo, from the PeopleCode Editor pop-up menu. Cut, Copy, and Paste use standard Microsoft Windows keyboard shortcuts. You can also cut, copy, and paste within the same PeopleCode program or across multiple programs.

Use these buttons to perform editing functions:

Save the current PeopleCode program. You can also use the key combination CTRL+S.

Cut the selected text or item. You can also use the CTRL+X or SHIFT+DEL key combinations.

Copy the selected text or item. You can also use the CTRL+C or CTRL+INS key combinations.

Paste from the clipboard. You can also use the CTRL+V or SHIFT+INS key combinations.

Find specified text. You can also use the key combination CTRL+F.

Find and replace specified text. You can also use the key combination CTRL+H.

Validate the current PeopleCode program.

Undo the last change. Use the CTRL+Z or ALT+BACKSPACE key combinations.

Cancel the current operation. Use ESC key.

Find and Replace Dialogs

When you use the Find and Replace functions, any text string that is highlighted appears when either the Find or Replace dialog boxes are called. For example, if you select the method ActiveRowCount it appears in the Find dialog box when it's called, as shown in the following example:

Find dialog box

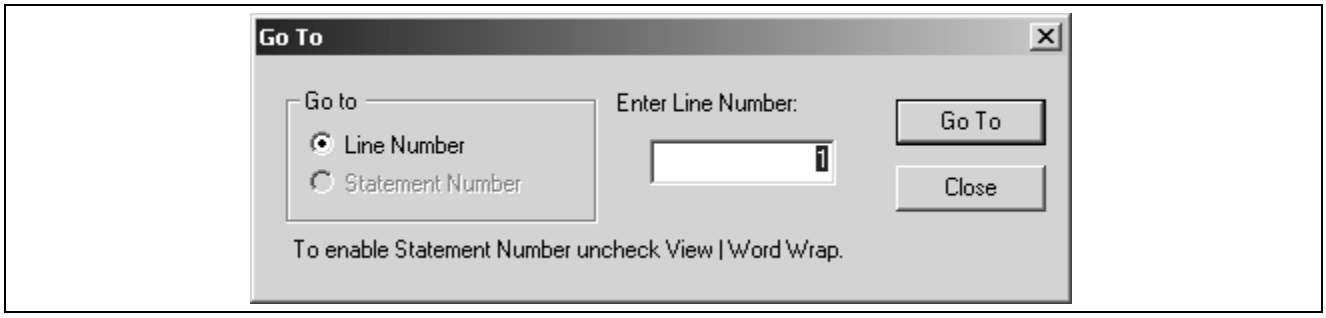
You can move through finding and replacing text strings one string at a time, or click Replace All to replace globally. The Undo function is available to undo the last replace or replace all.

The Mark All button places a bookmark next to all lines that have the matching text. Use SHIFT+CTRL+F2 to remove all bookmarks.

With the Replace dialog box, you can select to replace text either in a selected section or a whole file (that is, a PeopleCode program.)

Go To Dialog

Use the Go To dialog box to specify a line number in the current program, then go to that line. If you have line wrap not enabled, you can specify to go to statement numbers instead of line numbers.



Go To dialog box

Validate Syntax Utility

To check the syntax of the current PeopleCode program and format it if it is syntactically correct, do one of the following:

- Click the Validate Syntax button on the PeopleSoft Application Designer toolbar.
- Within PeopleSoft Application Designer, select Tools, Validate Syntax.
- Right-click in the PeopleCode Editor window, then select Validate Syntax.

The Validate utility has several functions, such as finding undeclared variables, mismatching data types, or invalid methods or properties for a class. You can check either a single component or an entire project.

Errors or warnings produced by the Validate utility are displayed in the Validate tab at the bottom of the PeopleCode Editor window.

Any variables that you don't declare are automatically declared for you, and a warning message appears in the Validate tab for each undeclared variable. You can right-click in the Validate tab and select Clear to delete all the warnings listed there, then use the Validate utility again to ensure that your code runs without errors or warnings.

Note. This feature is convenient if you have written multiple PeopleCode programs and you want to check the syntax of one without saving. All PeopleCode programs associated with an item (record, component, message definition, and so on) are checked prior to saving.

Short cut keys

The following table lists all the short cut keys available in the PeopleCode Editor. The short cut keys for the PeopleSoft Application Designer are not listed.

Key	Description
CTRL-A	Select all
CTRL-C	Edit copy
CTRL-F	Edit find
CTRL-H	Edit replace
CTRL-L	Line cut
SHIFT-CTRL-L	Line delete
CTRL-U	Selection lowercase
SHIFT-CTRL-U	Selection uppercase
CTRL-V	Paste
BACKSPACE	Backspace and delete characters
ALT-BACKSPACE	Edit undo
CTRL-BACKSPACE	Delete to start of word
SHIFT-ALT-BACKSPACE	Edit redo
DELETE	Delete
CTRL-DELETE	Delete to next word
SHIFT-DELETE	Edit cut
DOWN (arrow)	line down
CTRL-DOWN	Scoll window down one line
SHIFT-DOWN	Line down with selection
END	Position cursor at end of line

Key	Description
CTRL-END	Position cursor at end of file
SHIFT-END	Select to line end
SHIFT-CTRL-END	Select to end of file
ENTER	New line
ESC (escape)	clear selection
F2	Next bookmark
CTRL-F2	Toggle bookmark off and on
SHIFT-F2	Previous bookmark
SHIFT-CTRL-F2	Remove all bookmarks
F3	Find next
SHIFT-F3	Find previous
F5	Go (Debug)
F8	Step (Debug)
F9	Toggle debug breakpoint
ATL-F9	Edit breakpoints (Debug)
CTRL-F9	Break at start (Debug)
F10	Step over (Debug)
HOME	Position cursor to first character of line
CTRL-HOME	Position cursor to start of file
SHIFT-HOME	Select to start of line
SHIFT-CTRL-HOME	Select to start of file
INSERT	Toggle insert mode
CTRL-INSERT	Copy
SHIFT-INSERT	Paste
LEFT (arrow)	Position cursor left one character
CTRL-LEFT	Posistion cursor left one word
SHIFT-LEFT	Select one character left of cursor

Key	Description
SHIFT-CTRL-LEFT	Select next word left of cursor
PAGE DOWN	Page down
PAGE UP	Page up
RIGHT(arrow)	Position cursor right one character
CTRL-RIGHT	Position cursor right one word
SHIFT-RIGHT	Select one character right of cursor
SHIFT-CTRL-RIGHT	Select next word right of cursor
TAB	Tab
SHIFT-TAB	Back tab
UP(arrow)	Line up
CTRL-UP	Scroll window up one line
SHIFT-CTRL-W	Select word
CTRL-X	Edit cut
CTRL-Y	Edit redo
CTRL-Z	Edit undo
SHIFT-CTRL-Z	Edit redo

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer, "Working With Projects," Validating Projects

Chapter 16, "Debugging Your Application," Compiling All PeopleCode Programs at Once, page 263

Formatting Code Automatically

You do not need to format your PeopleCode statements; you need only to use the correct syntax. When you save or validate, the system formats the code according to the rules in the PeopleCode tables, no matter how you entered it originally. The PeopleCode Editor automatically converts field names to uppercase and indents statements.

PeopleCode is case-insensitive, except for quoted literals. PeopleCode does not format anything surrounded by quotation marks. String comparisons, however, are case-sensitive. When you compare the contents of a field or a variable to a string literal, make sure the literal is in the correct case.

All field names in a PeopleCode program must be fully qualified, even if the field is on the same record definition as the PeopleCode program. However, you only need to type in the name of the field. The editor validates if the field exists on the current record, and reformats the fieldname to *recordname.fieldname*.

Using Drag-and-Drop Editing

In addition to the standard keyboard shortcuts and toolbar buttons, you can copy or move text within a window or between two PeopleCode Editor windows by using the mouse and the CTRL key.

Note. You cannot open two editor windows for a single parent definition, or for any two of its child definitions.

To move text between instances of the PeopleCode Editor:

1. Select the text you want to move.
2. Place the mouse over the text and drag the text to the other PeopleCode Editor window.
3. When the cursor appears at the place where you want to insert the text, release the mouse button.

To copy text between instances of the PeopleCode Editor:

1. Select the text you want to move.
2. Hold down the CTRL key as you drag the text to the other PeopleCode Editor window.
3. When the cursor appears at the place where you want to insert the text, release the mouse button.

Accessing PeopleCode External Functions

An external PeopleCode function is a function written in PeopleCode (as opposed to a built-in function or external DLL function) and defined in a program outside the one from which it is called. External PeopleCode functions can be defined in any record PeopleCode program, but typically they are stored in the FieldFormula event in records beginning with *FUNCLIB_*.

The PeopleCode Editor provides immediate access to external PeopleCode function definitions. Right-click the function name in the program where the function is called, then select View Function *FunctionName*. This opens a new PeopleCode Editor window containing the external function definition.

Note. Internet scripts are contained in records similar to FUNCLIB_ records. However, their names begin with *WEBLIB_*.

Accessing Definitions and Associated PeopleCode

You can open a field, record, page, message, and other definitions from the PeopleCode Editor. Or you can open a new PeopleCode Editor window containing the programs associated with a field, record, page, or other definition.

To open a definition from the PeopleCode Editor, right-click a PeopleCode definition reference and select View Definition.

For example, you could open definitions by clicking the following references:

- Record.BUS_EXPENSE_PER
- BUS_EXPENSE_PER.EXPENSE_PERIOD_DT
- Page.BUSINESS_EXPENSES

If you access a record definition from a record field reference (that is, *recordname.fieldname*) the specified record field is selected when the record definition opens.

To open a new PeopleCode editor window, right-click a reference to the definition and select View PeopleCode.

For example, you can access record PeopleCode from the following record and record field references:

- Record.BUS_EXPENSE_PER
- BUS_EXPENSE_PER.EXPENSE_PERIOD_DT

Note. You can only view the PeopleCode and definition when the text is in the format *recordname.fieldname.* If the text is in the format *method(i).recordname, method(i).fieldname,* or *&MyRecord.Fieldname,* the View PeopleCode and View Definition commands are not available.

Accessing Help

The PeopleCode Editor has context-sensitive online help for all PeopleCode built-in functions, methods, properties, system variables, and meta-SQL. To access online help, place the cursor in the name of what you want to look up, then press F1. If there is a corresponding entry in the online reference system it appears; otherwise a No Help Available error message appears.

If more than one entry is applicable, a pop-up window that lists all applicable entries appears. Select the correct entry.

See Also

Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," PeopleTools Options

Setting up Help

To set up the help, use the F1 Help URL field on the PeopleTools Options page to specify where the documentation is stored.

The format of the URL is as follows:

http://*doc_location***/f1search.htm?ContextID=%CONTEXT_ID%&LangCD=%LANG_CD%**

The *doc_location* specifies where the documentation files are located on your system. The rest of the URL exactly the above format.

For example, you might place the following URL in the F1 Help URL field:

http://Pandora/doc/f1search.htm?ContextID=%CONTEXT_ID%&LangCD=%LANG_CD%

After you specify the help location, you must exit all PeopleTools sessions and start again before you can access the help.

Changing Colors in the PeopleCode Editor

You can change the display (foreground) color for many language elements in the PeopleCode Editor, including quoted strings, keywords, and built-in functions. You can also change the background color.

To change the display colors:

1. Select Edit, Display Font and Colors.
2. Select the language element that you want to change.
3. Select the foreground color.

   If you click the Automatic checkbox, the default color is used.

A box displaying the selected color is only available if the Automatic checkbox is not selected.

If you click the box displaying the selected color, the standard color chart for your display appears. If you click Other from this dialog, or click the drop down button from the Font and Color Settings dialog box, the custom color chart for your display appears.

4. Select the background color.

If you click Reset All, the default colors for the PeopleCode language elements are reassigned.

Selecting a Font for the PeopleCode Editor

The default font for the PeopleCode Editor is 9-point Courier New.

To change the PeopleCode Editor font, select Edit, Display Fonts and Colors. Use this dialog box to change the font for the editor.

Note. When you select a font for the PeopleCode Editor, the font selection dialog box provides choices based on a character set appropriate for your international version of Microsoft Windows. If you experience trouble embedding foreign characters (such as Thai characters) in PeopleCode, you might need to change the font setting. If you are trying to display Thai characters in Microsoft Windows 95, you might also need to change your keyboard input settings for the characters to display correctly. You can change your keyboard input settings from the Input Locales tab on the Windows Regional Settings control panel, or on the Keyboard control panel.

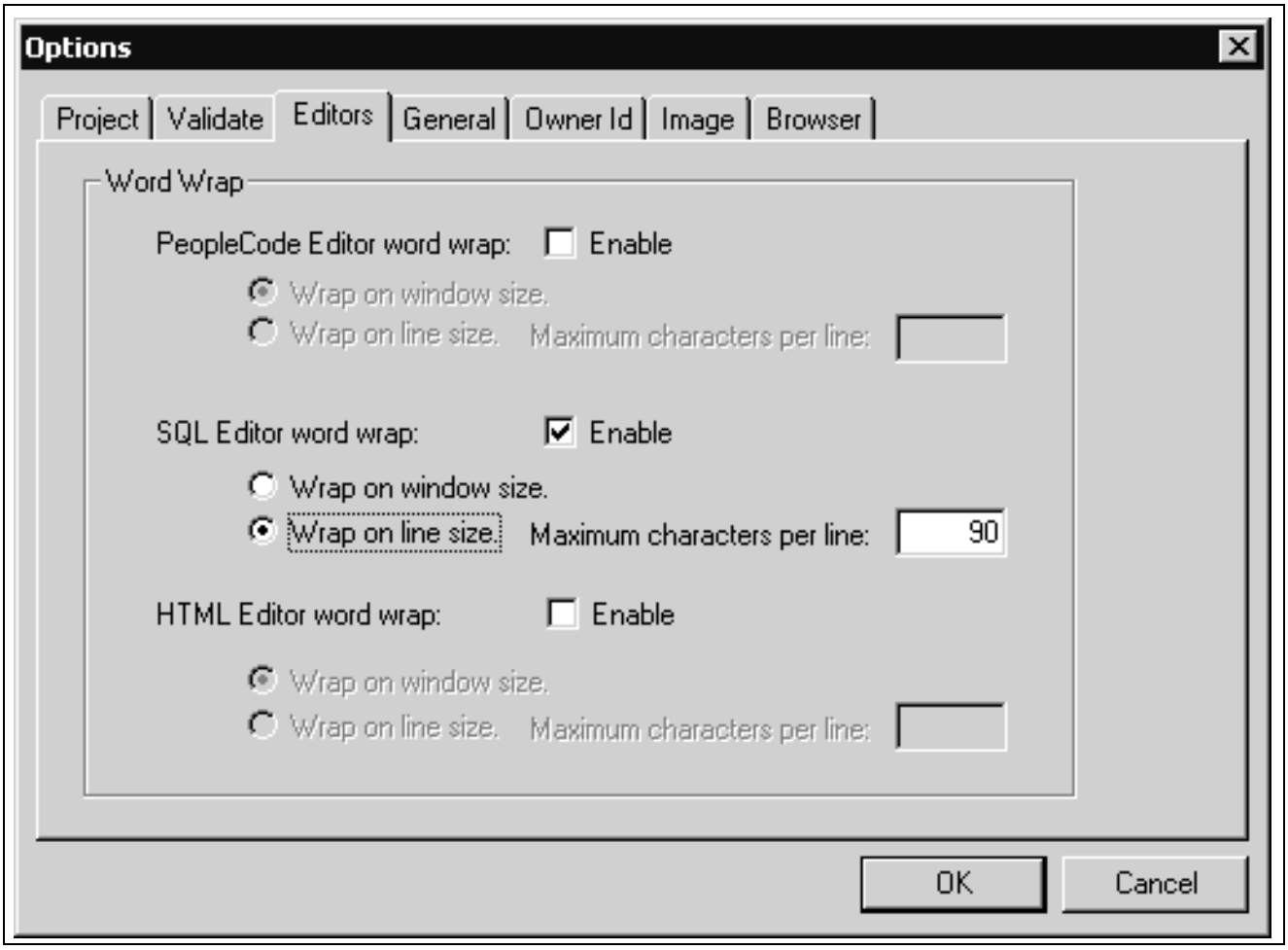
Changing Word Wrap in the PeopleCode Editor

The PeopleCode Editor supports text word wrapping. You can turn word wrapping on and off for an open editor window. You can also specify the default value for word wrap, as well as whether the text wraps to the editor's window size or to a fixed number of characters per line.

To turn word wrapping on or off for an open editor window, go to Edit, Word Wrap. After you close PeopleSoft Application Designer, all word wrap values are reset to the default value for the editor.

Specifying Word Wrap Options

Go to the Tools, Options dialog box, Editors tab, to specify the word wrap options.



Options dialog box: Editors tab

Enable (word wrap)	Specify whether word wrap is the default mode when opening the editor. If this box is not checked, wrapping text based on window size is the default.
Wrap on Window Size	Specify whether the text wraps based on the size of the window.
Wrap on Line Size	Specify whether the text wraps based on the number of charaters in a line. If this box is checked, you can specify the number of maximum number of characters per line.
Maximum Characters per Line	Specify the maximum number of characters allowed for a line before the text wraps. The default value is 90. Valid values are between 25 and 2000.

Using the PeopleCode Event Properties

To access the PeopleCode Event properties, open a PeopleCode editor window, then either press ALT + ENTER or click the Properties button.

PeopleCode Event Properties dialog box

Note. This dialog box has been deprecated. It has no effect on the location of the execution of code.

Generating PeopleCode Using Drag-and-Drop

You can generate references to definitions using a drag-and-drop operation. You can also generate PeopleCode templates for accessing business interlinks and component interfaces.

This section discusses how to:

- Generate definition references.
- Generate PeopleCode for a business interlink.
- Generate PeopleCode for a component interface.
- Generate PeopleCode for a file layout.

Generating Definition References

When you drag definitions, such as menus, records, record fields, and pages, from a project into an open PeopleCode editor window, you generate a reference to the definition. For example, suppose your project contain a component named QEACTIVITY_GUIDE_1. If you drag the QEACTIVITY_GUIDE_1 component definition from the project into an open PeopleCode window, the word QEACTIVITY_GUIDE_1 prefixed with the keyword **COMPONENT** is written to the PeopleCode program in the place where you dragged the definition. Similarly if you dragged a message definition, the name of the message definition would be written to the PeopleCode program, prefaced with the keyword **MESSAGE**.

Generating PeopleCode for a Business Interlink

After you create a business interlink definition, you use PeopleCode to instantiate an interlink object and activate the interlink plug-in. This PeopleCode can be long and complex. Rather than write it directly, you can drag and drop the business interlink definition from the PeopleSoft Application Designer Project view into an open PeopleCode edit pane. PeopleCode Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify to suit your purpose.

The following is a snippet of the code that is generated:

```
/* ===>
/* ===>
 This is a dynamically generated PeopleCode template to be
```

```
used only as a helper to the application developer.
You need to replace all references to '<*>' OR default values
with  references to PeopleCode variables and/or a Rec.Fields.*/

/* ===> Declare and instantiate: */
Local Interlink &QE_AE_NONSSL__1;
Local BIDocs &inDoc;
Local BIDocs &outDoc;
Local boolean &RSLT;
Local number  &EXECRSLT;
&QE_AE_NONSSL__1 =  GetInterlink(INTERLINK.QE_AE_NONSSL_BI);
.
.
.
.
```

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Business Interlinks, "Getting Started with PeopleSoft Business Interlinks"

Generating PeopleCode for a Component Interface

After you create a component interface definition, you can use PeopleCode to access it. This PeopleCode can be long and complex. Rather than write it directly, you can drag and drop the component interface definition from the PeopleSoft Application Designer Project view into an open PeopleCode edit pane. PeopleSoft Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify to meet your requirements.

The following is a snippet of the code that is generated:

```
/* ===>
This is a dynamically generated PeopleCode template to be
used only as a helper to the application developer.
You need to replace all references to '<*>' OR default values
with references to PeopleCode variables and/or a Rec.Fields. */


Local ApiObject &oSession;
Local ApiObject &oCurrencyCdCi;
Local ApiObject &oPSMessageCollection;
Local ApiObject &oPSMessage;
Local File &LogFile;
Local number &i;
Local String &strErrMsgSetNum, &strErrMsgNum, &strErrMsgText,
&strErrType;
.
.
.
```

You can also access a component interface using the component object model (COM). You can automatically generate a Visual Basic template, a Java template, or a C template, similar to the PeopleCode template, to begin.

To generate a template:

1. Open a component interface in PeopleSoft Application Designer.
2. Right-click anywhere in the open component interface and select a template type.

   You must save the component interface before generating the template.

   When the template is successfully generated, a message appears with the full path and name of the file containing the template.
3. Open the generated file and modify the source code to meet the needs of your application.

   The following is the initial code snippet that is generated for a Visual Basic template:

```
Option Explicit
'===>
'This is a dynamically generated Visual Basic template to be
'used only as a helper to the application developer.
'You need to replace all references to '<*>' OR default
'values with  references to Visual Basic variables.

Dim oSession As PeopleSoft_PeopleSoft.Session

Private Sub ErrorHandler()
 '***** Display PeopleSoft Error Messages *****
 If Not oSession Is Nothing Then
  If oSession.ErrorPending Or oSession.WarningPending Then
   Dim oPSMessageCollection As PSMessageCollection
   Dim oPSMessage As PSMessage
   Set oPSMessageCollection = oSession.PSMessages
   Dim i As Integer
   For i = 1 To oPSMessageCollection.Count
    Set oPSMessage = oPSMessageCollection.Item(i)
    Debug.Print "(" & oPSMessage.MessageNumber & "," &
oPSMessage.MessageSetNumber & ") : " & oPSMessage.Text
   Next i
   '***** Done processing messages in the collection;
   '***** OK to delete *****
   oPSMessageCollection.DeleteAll
  End If
 End If
End Sub
.
.
.
.
```

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Business Interlinks, "Getting Started with PeopleSoft Business Interlinks"

Generating PeopleCode for a File Layout

After you create a file layout definition, you can use PeopleCode to access it. This PeopleCode can be long and complex. Rather than write it directly, you can drag and drop the file layout definition from the PeopleCode Application Designer Project view into an open PeopleCode edit pane. PeopleSoft Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify to meet your requirements.

The following is a snippet of the code that is generated:

```
Function EditRecord(&REC As Record) Returns boolean ;
Local integer &E;

REM   &REC.ExecuteEdits(%Edit_Required + %Edit_DateRange +
%Edit_YesNo + %Edit_TranslateTable + %Edit_PromptTable +
%Edit_OneZero);

   &REC.ExecuteEdits(%Edit_Required + %Edit_DateRange +
%Edit_YesNo + %Edit_OneZero);
   If &REC.IsEditError Then
      For &E = 1 To &REC.FieldCount
         &MYFIELD = &REC.GetField(&E);
         If &MYFIELD.EditError Then
            &MSGNUM = &MYFIELD.MessageNumber;
            &MSGSET = &MYFIELD.MessageSetNumber;
            &LOGFILE.WriteLine("****Record:" | &REC.Name | ",
Field:" | &MYFIELD.Name );
            &LOGFILE.WriteLine("****" | MsgGet(&MSGSET,
&MSGNUM, ""));
         End-If;
      End-For;
      Return False;
   Else
      Return True;
   End-If;
End-Function;
.
.
.
.
```

CHAPTER 14

Using the SQL Editor

This chapter provides an overview of the Structured Query Language (SQL) Editor window and discusses how to:

- Access SQL definition properties.
- Access the SQL editor.
- Use the SQL editor.

See Also

Chapter 13, “Using the PeopleCode Editor,” page 219

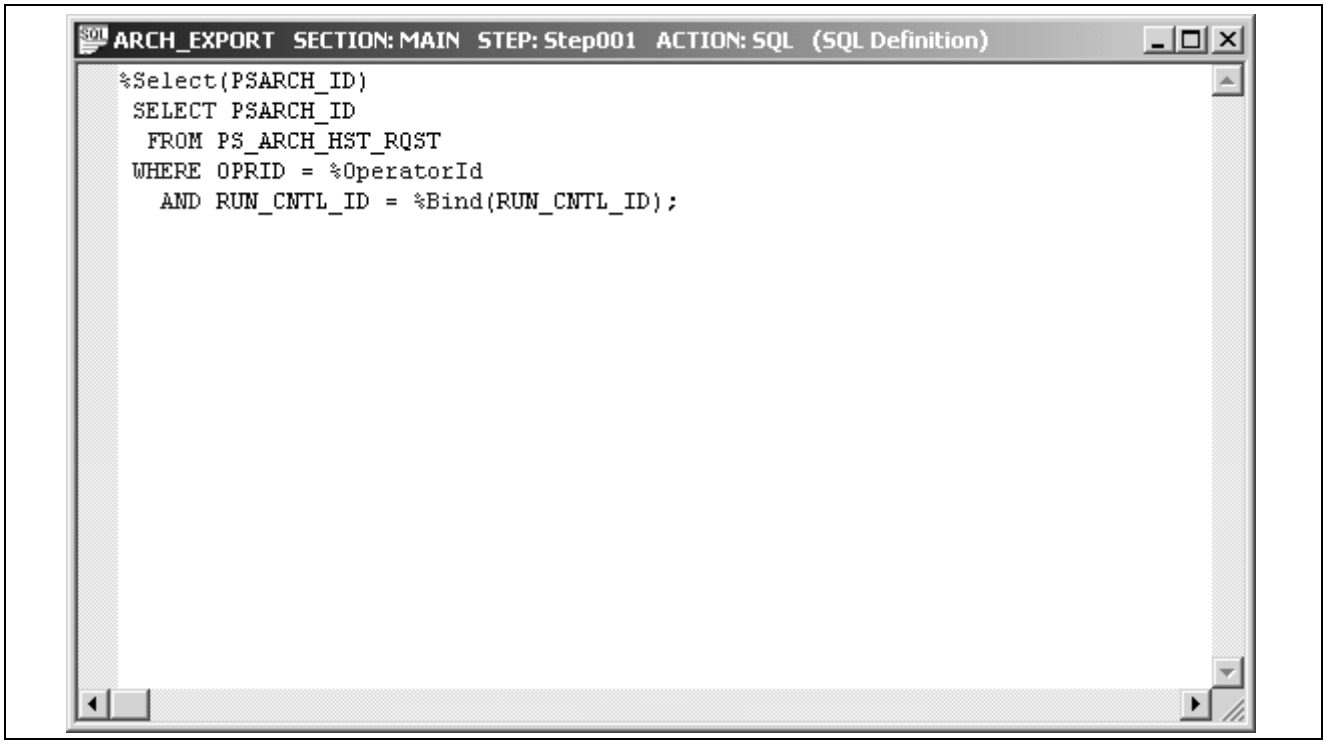
Understanding the SQL Editor Window

Use the SQL Editor to create SQL for SQL definitions, record views, and Application Engine programs.

The SQL Editor and the PeopleCode editor interfaces are similar. You can add, delete, and change text: you can use the find and replace function; and you can validate the SQL. When you save a SQL definition, the code is automatically formatted (indented and so on), the same as it is for a PeopleCode program. You can select the colors for displaying keywords, comments, operators and so on. You can also specify word wrap options.

See Chapter 13, “Using the PeopleCode Editor,” Using the PeopleCode Editor, page 222.

The editor window’s title bar displays either the name of the SQL definition or the name of the component that contains the SQL. For example, if the SQL statement is part of an Application Engine program, the names of the program, the section, the step, and the action are listed in the title bar, as shown in the following example:



SQL Editor window with Application Engine program SQL

The editor window consists of the main edit pane. For SQL definitions and SQL used with records, there is also a drop-down database list at the upper left. For SQL definitions, you can also use a drop-down effective date list at the upper right.

Note. When you make a selection from either drop-down list box, your selected entry has a yellow background, indicating that you must click the edit pane before you can start typing.

Accessing SQL Definition Properties

Access the definition properties for the SQL definition by either:

- Pressing ALT+ENTER.
- Selecting File, Definition Properties.
- Right-clicking in the definition and selecting Definition Properties.

Use general properties to specify a description for the SQL definition as well as additional comments. The description appears in PeopleSoft Application Designer search lists.

Use the advanced properties to display an effective date with the SQL definition.

Note. The Audit SQL field on the Advanced Properties tab is not used.

Accessing the SQL Editor

This section discusses how to:

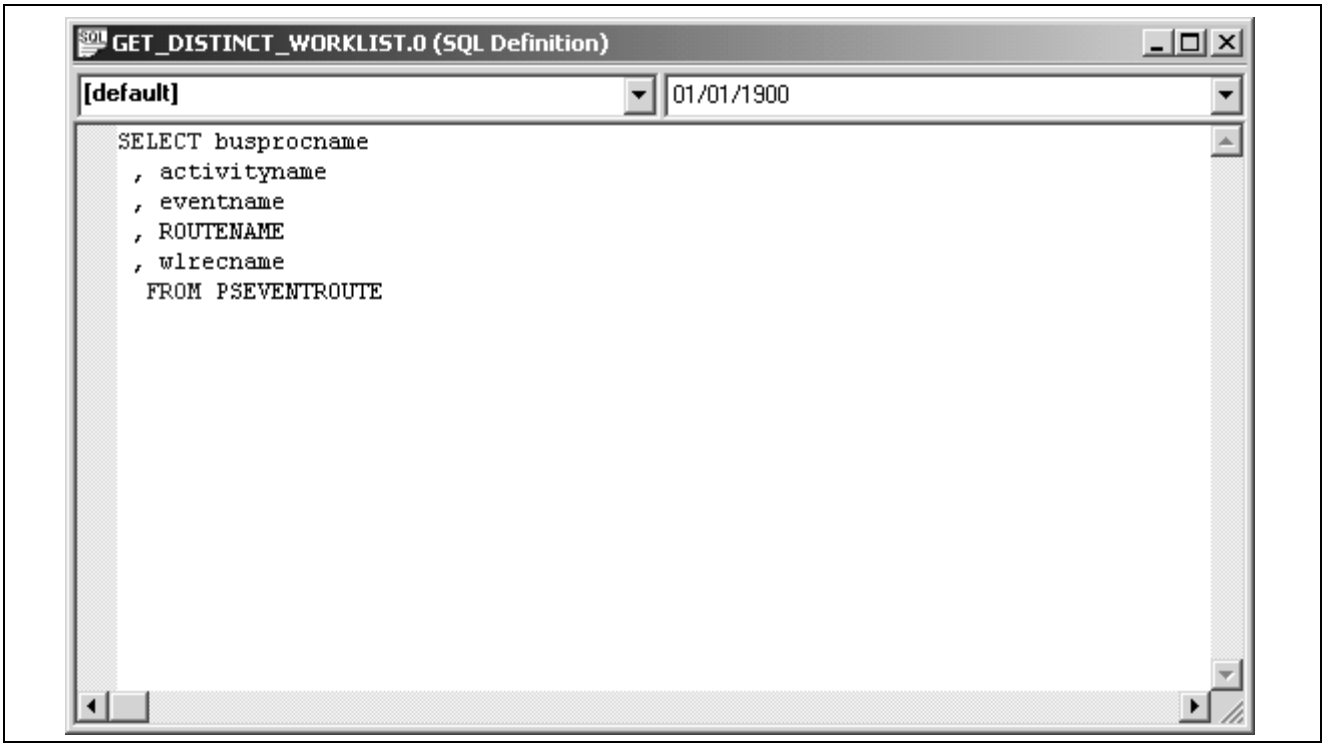
- Create SQL definitions.

- Create dynamic view or SQL view records.
- Access the SQL editor from Application Engine programs.

You access the SQL editor differently for each type of component.

Creating SQL Definitions

A SQL definition contains SQL statements, which can be entire SQL programs or just fragments that you want to reuse. You can access, create, change, or delete SQL definitions using PeopleSoft Application Designer, or you can use the SQL class in PeopleCode. SQL definitions are upgradable, and you can add them to a project. The following illustration shows a SQL definition:



SQL definition with effective date

To create a SQL definition:

1. From PeopleSoft Application Designer, select File, New, SQL.
2. Specify the database type to associate with the SQL definition.

   You can associate more than one database type with a single SQL definition. In PeopleCode, you can specify the appropriate database type for the program. However, at least one of the SQL statements must be of type Default.
3. (Optional) Specify an effective date.

   To specify an effective date with your SQL definition:

   a. Access the object properties by selecting File, Object Properties.

      You can also select the SQL definition, right-click and then select Object Properties, or press ALT + ENTER.

   b. Click the Advanced tab, then click Show Effective Date.

      When you click OK, the SQL definition shows a date in the right-hand drop-down menu.

4. Add the SQL code.

   You do not need to format your code. The SQL editor formats it when you save the SQL definition.

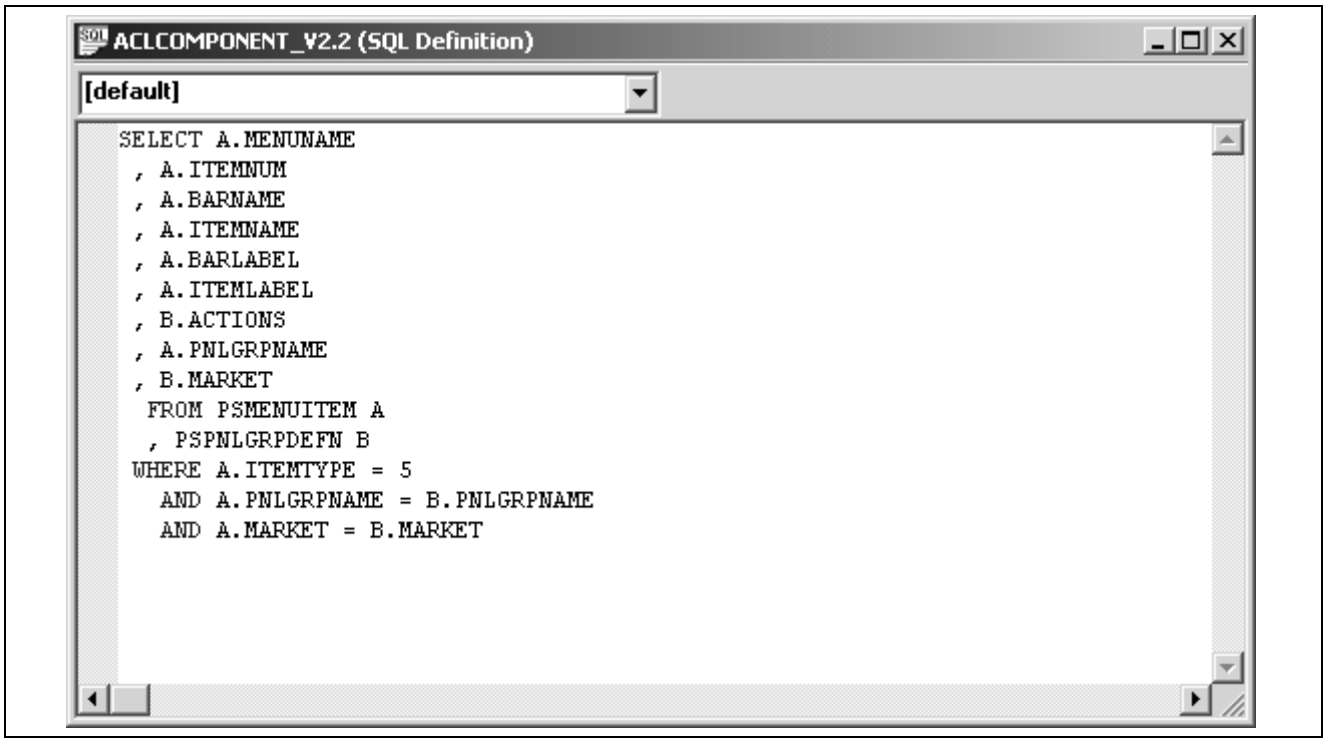
See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer, "Using PeopleSoft Application Designer"

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

Creating Dynamic View or SQL View Records

When you create a SQL view or dynamic view record definition, you enter a SQL view Select statement to indicate the field values that you want to join and the tables that contain the field values. You do this in the SQL editor, as shown in the following example:



SQL Editor for SQL view record definition

To access the SQL editor with records:

1. Open or create a dynamic view or SQL view record definition.
2. Select the Record Type tab.
3. Click the Click to Open SQL Editor button.

   You can select a database type, but not an effective date, from the SQL editor for dynamic view and SQL view record definitions.

Note. You must be sure to save record definitions of the SQL View type prior to opening the SQL Editor. Once the SQL Editor is open, the Save options are disabled and inaccessible. If you do not save your changes before opening the SQL Editor, you may lose your work.

Considerations with View SQL and Union Statements

If you've selected the Platform Compatibility Mode check box under PeopleTools, Utilities, PeopleTools Options you can not use Union statements in a view. If you try to specify a Union Select statement, you won't be able to save the SQL definition.

See Also

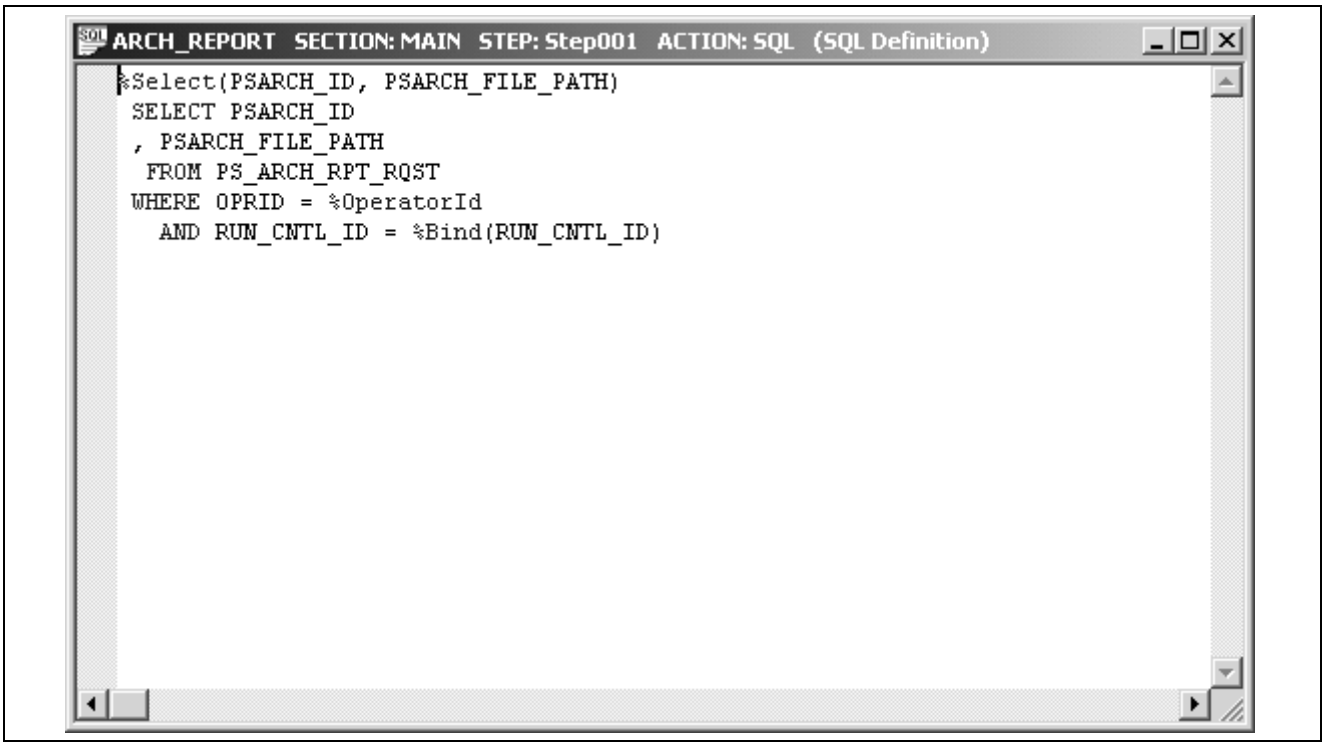
Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," PeopleTools Options

Accessing the SQL Editor from Application Engine Programs

You can access the SQL editor from the following action types:

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

The following example shows an Application Engine program in the SQL Editor:



Application Engine SQL Editor window

To access the SQL editor in an Application Engine program:

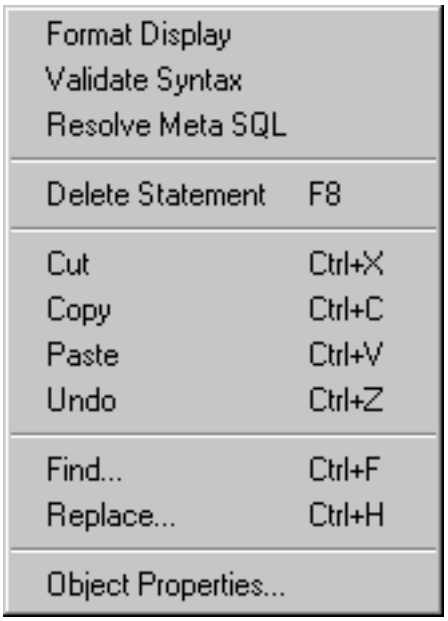
1. Open the Application Engine program.
2. Select the action.
3. Either right-click and select View SQL, or select View, SQL.

Select the database type and effective date for this SQL in the section, not in the SQL editor.

Using the SQL Editor

The SQL editor works similarly to any other text editor. You can use the same functions as with the PeopleCode editor: cut, paste, find, replace, and so on.

When you right-click in an open SQL editor window, you see available functions for the SQL editor:



SQL Editor shortcut menu

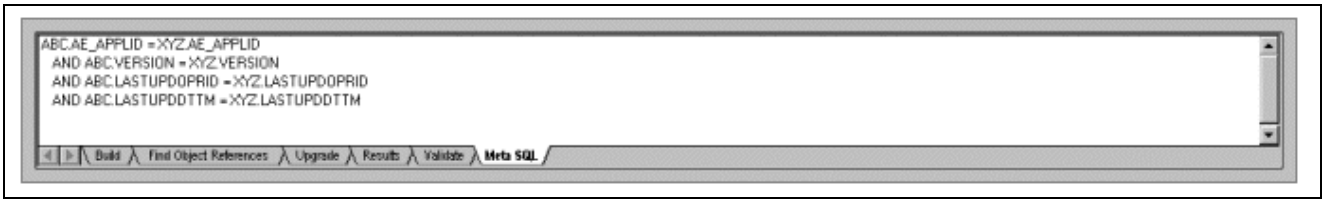
The following functions are available for the SQL editor, but are not available for the PeopleCode editor.

Function	Description
Format Display	You do not need to format your SQL statements; you need to use the correct syntax only. When you save or validate, the system formats the code according to the rules in the PeopleCode tables, no matter how you entered it originally. It automatically converts field names to uppercase and indents statements for you. The SQL then looks consistent with other programs in the system.

Function	Description
Resolve Meta-SQL	If there is meta-SQL in the SQL, select Resolve Meta-SQL to expand the meta-SQL statement in the output window, under the Meta-SQL tab.
Delete Statement	You can delete standalone SQL statements. This menu item is not enabled with SQL statements that have a database type of Default with no effective date, or for statements that have a database type of Default and an effective date of 01/01/1900.

For example, using Resolve Meta-SQL, the following code expands as follows:

```
%Join(COMMON_FIELDS, PSAEAPPLDEFN ABC, PSAESECTDEFN XYZ)
```



Meta-SQL expanded in the output window

See Also

Chapter 13, “Using the PeopleCode Editor,” Writing and Editing PeopleCode, page 223

CHAPTER 15

Creating Application Packages and Classes

This chapter provides an overview of application packages and discusses how to:

- Create application packages.
- Use the Application Package Editor.
- Edit application package classes.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference, "Application Classes"

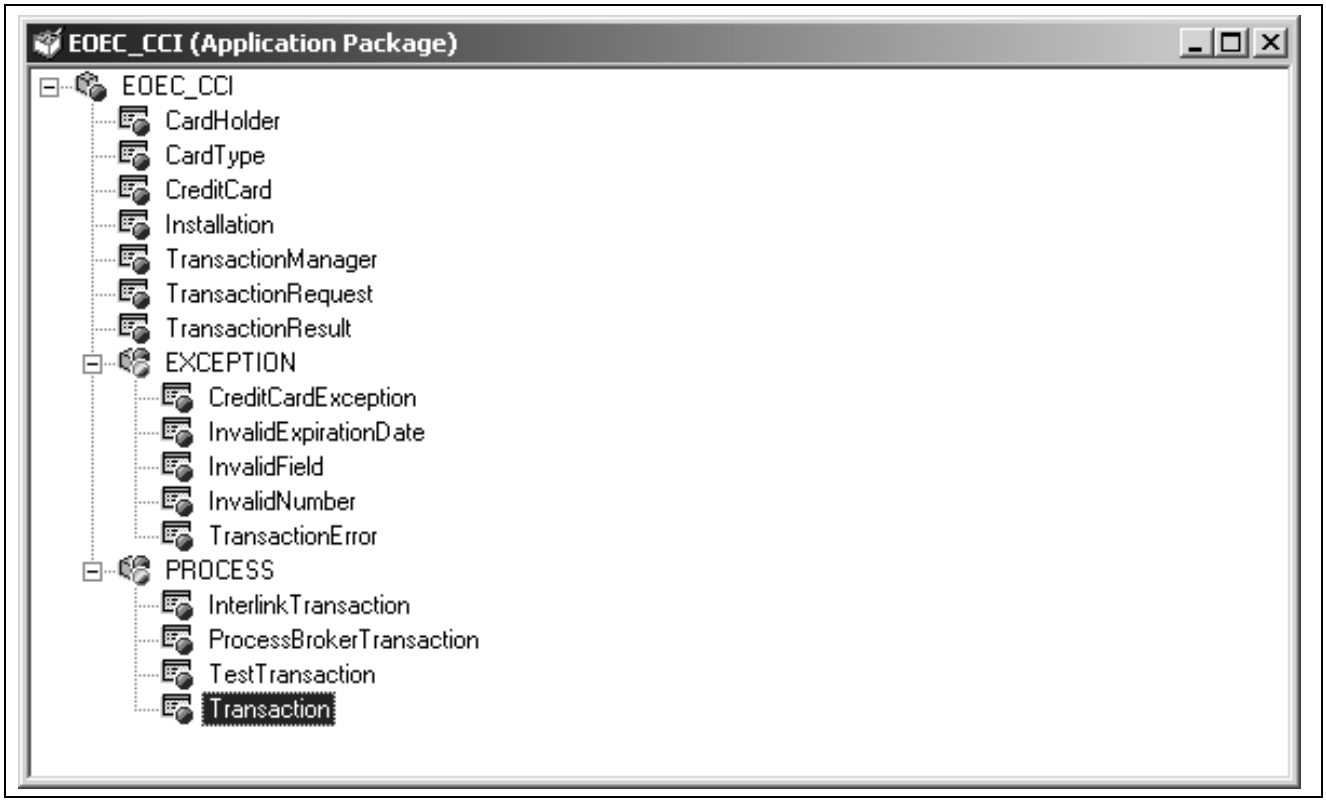
Understanding Application Packages

Use the Application Packages Editor to create application packages. A *package* contains other packages or application classes. A *subpackage* is any package within a primary, or parent, package.

The editor window's title bar displays the name of the application package definition. The main window displays the classes and other application packages that make up the application package definition.

The primary package is represented by a blue database symbol, while subpackages are represented with yellow database symbols. Classes are represented by lightning bolts and scripts, and host-related application object classes (PeopleCode programs).

In the following example, EOEC_CCI is the primary package, and EXCEPTION and PROCESS are subpackages. CardHolder, CareType, CreditCard, and so on, are classes in the EOEC_CCI application package, while CreditCardException and InvalidExpirationDate are classes in the EXCEPTION subpackage, and InterlinkTransaction and TestTransaction are classes in the PROCESS subpackage.



Application Package Editor main window

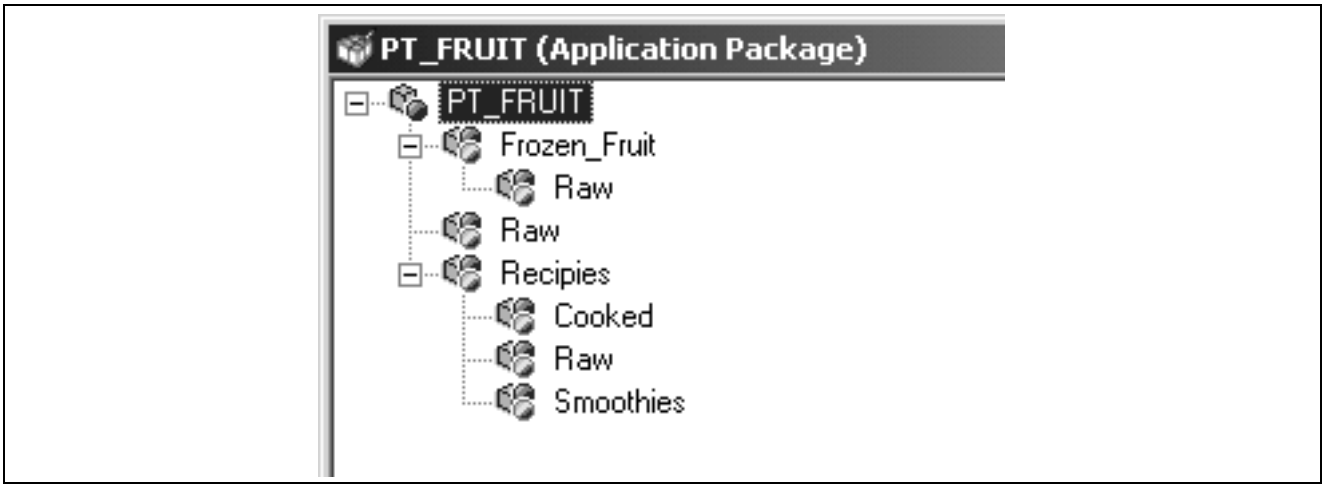
Creating Application Packages

This section provides an overview of package names and discusses how to create application packages.

Understanding Package Names

You can create a subpackage with the same name as another package or subpackage within the same application package definition, as long as the fully qualified name is unique for each subpackage. Each subpackage is differentiated by the full path name of the class (from the package definition name and the subpackage name).

For example, suppose in the application class PT_FRUIT, where PT_FRUIT is the primary class, you had the following structure of subpackages (no classes are listed in this example):



Example of application package naming conventions

In this example, there are three subpackages named Raw, but the fully qualified name for each is unique. For example, the first one is qualified by the name of the primary package. Its fully qualified name is PT_FRUIT:Raw.

The other Raw subpackages are also qualified by the subpackages that contain them. Their names are PT_FRUIT:Reciepies:Raw and PT_FRUIT:Smoothies.Raw.

Similarly, you cannot create two classes with the same name within a given package or subpackage. You can create classes with the same name within the same application package definition, just like subpackages, as long as the fully qualified name is unique. Each class is differentiated by the full path name of the class.

Note. You cannot create a structure where there are more than two levels of subpackages defined below the primary package.

Creating Application Package Definitions

To create a new application package:

1. Open PeopleSoft Application Designer.
2. Select File, New, Application Package.

   A new application package appears.

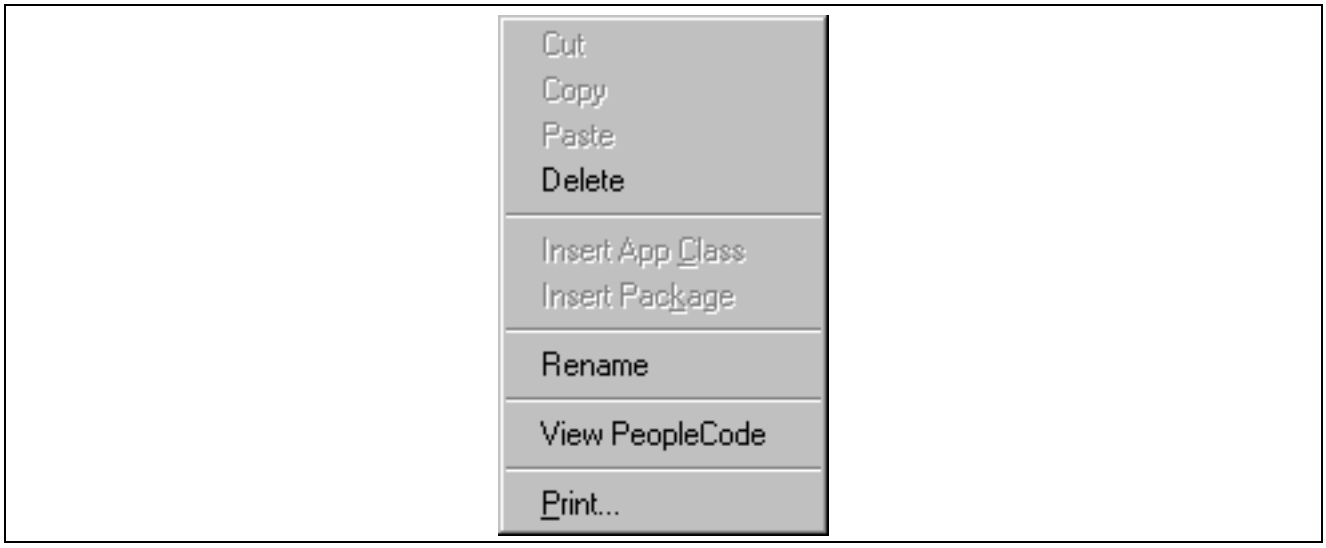
To insert a new package or class:

1. Open an application package.
2. Select a package.
3. Insert the new item.

   Select the package, then select Insert, Package, or Insert, Application Class.

Using the Application Package Editor

When you right-click an open Application Package Editor window, you see the available functions:



Application Package Editor shortcut menu

Cut, Copy, and **Paste** — Not available for this release. You can work around this by inserting new subpackage and class nodes where needed and using the clipboard to copy and paste PeopleCode text from class to class.

To copy the primary package, select File, Save As.

Delete — Deletes either a class or a package. The PeopleCode text is not actually deleted until you save the application package. Deleted PeopleCode classes can be recovered by reinserting the class node, as long as you haven't saved in the interim.

Insert App Class (insert application class) — Inserts an application class. Because classes cannot have children (subclasses), they can be inserted only into an existing package.

Insert Package — Inserts an application package. You can only insert packages into an existing package or subpackage.

Rename — Renames either a class or a subpackage. When you save the definition, all PeopleCode programs associated with the renamed class are also updated. To rename the primary package definition, use File, Rename.

View PeopleCode — View the associated PeopleCode. PeopleCode can be only defined for application classes, and is not directly related to package nodes.

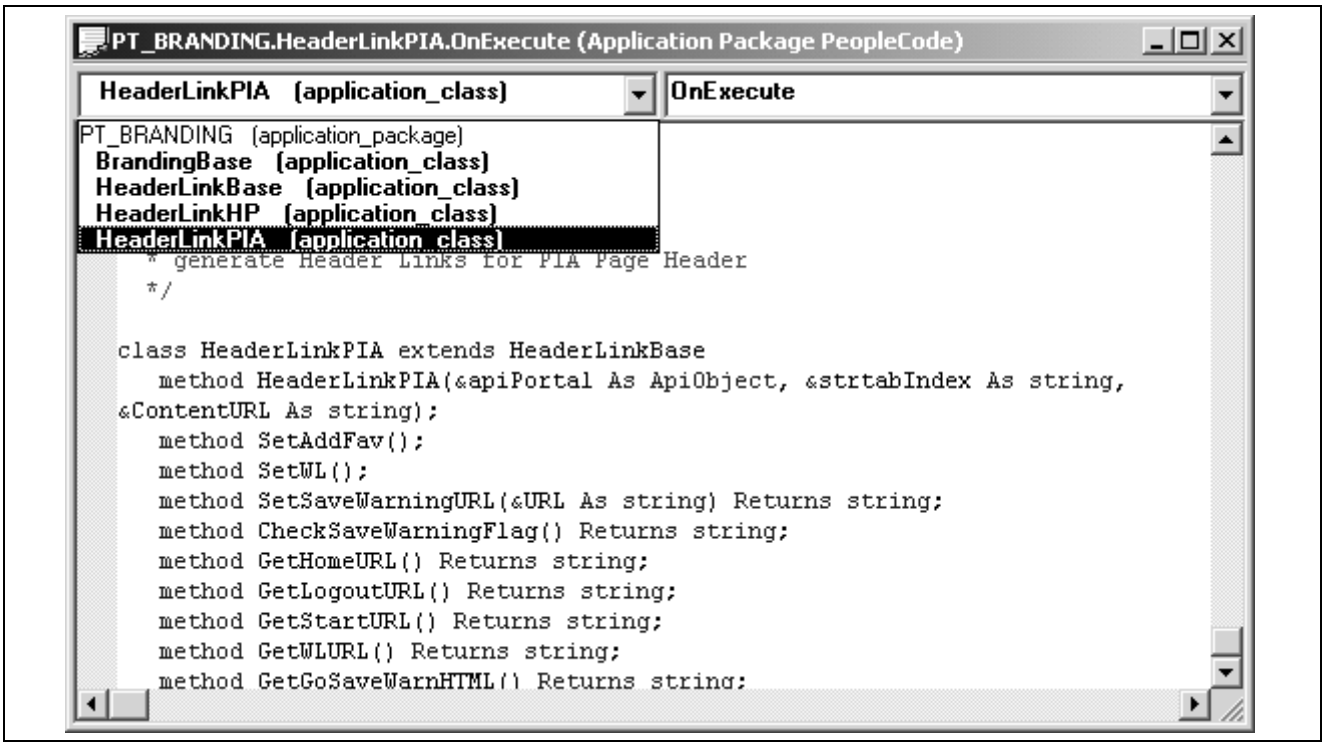
Print — Print the application package definition, including all the PeopleCode in the classes.

Editing Application Package Classes

From an application package, you can access the PeopleCode programs associated with the classes of the package.

The Application Packages Editor and the PeopleCode Editor interfaces are similar. You can add, delete, and change text: you can use the find and replace function; you can validate the syntax. When you save application packages, the code is automatically formatted (indented and so on), just as it is in the PeopleCode Editor.

The editor window contains the main edit pane, the drop-down definition list at the upper-left, and the drop-down event list at the upper-right, as shown in the following illustration:



Application Packages Editor window

Only one event is defined for an application class, OnExecute. This is not an event in the Component Processor flow. The application class runs when called.

The drop-down list on the upper-left enables you to navigate directly to the PeopleCode associated with every class in the package, as well as every subpackage and its classes.

To edit an application class:

1. Open the application package.
2. Select a class.
3. Either select View, PeopleCode, or right-click and select View PeopleCode.

   A PeopleCode Editor window appears.

See Also

Chapter 13, “Using the PeopleCode Editor,” Writing and Editing PeopleCode, page 223

CHAPTER 16

Debugging Your Application

This chapter provides an overview of the PeopleCode debugger and discusses how to:

- Access the PeopleCode debugger.
- Understand PeopleCode debugger features.
- Understand PeopleCode debugger options.
- Set up the debugging environment.
- Debug subscription PeopleCode.
- Compile PeopleCode programs at once.
- Set PeopleCode debugger log options.
- Interpret the PeopleCode debugger log file.
- Use the Find In feature.
- Use cross-reference reports.

Understanding the PeopleCode Debugger

The PeopleCode debugger is an integrated part of PeopleSoft Application Designer. The interface to the debugger has a visual indicator of breakpoints, an arrow indicating the current line, and the ability to step through code. You can inspect the value of a variable by holding the cursor over it and reading the pop-up bubble help. The debugger also provides variable inspection windows for global variables, local variables, function parameters, and component-scoped variables. It also enables PeopleCode objects to be expanded, so you can inspect their component parts.

Note. The PeopleCode debugger does not work on Microsoft Windows 95 or Windows 98.

Accessing the PeopleCode Debugger

You access the debugger through PeopleSoft Application Designer. You can start a debugging session either before or after you start a PeopleSoft component.

To start the PeopleCode debugger:

1. Open PeopleSoft Application Designer.
2. Select Debug, PeopleCode Debugger Mode.

   The Local Variables watch window opens. PeopleCode programs that had breakpoints set from your previous debugging session are opened also, and the breakpoints are restored.

Note. If you have already opened the debugger, then closed it, the menu may not change correctly to enable you to access the debugger a second time. If this occurs, click the Local Variables window, then try the Debug menu again.

You must make some adjustments to the system to run the debugger in PeopleSoft Pure Internet Architecture or with application message subscription PeopleCode.

Note. Your security administrator has options for allowing users to access different parts of PeopleSoft Application Designer, including the PeopleCode debugger. If you're having problems accessing the debugger, you may need to contact your system administrator concerning your security access. It is possible to access the PeopleCode debugger from outside a firewall.

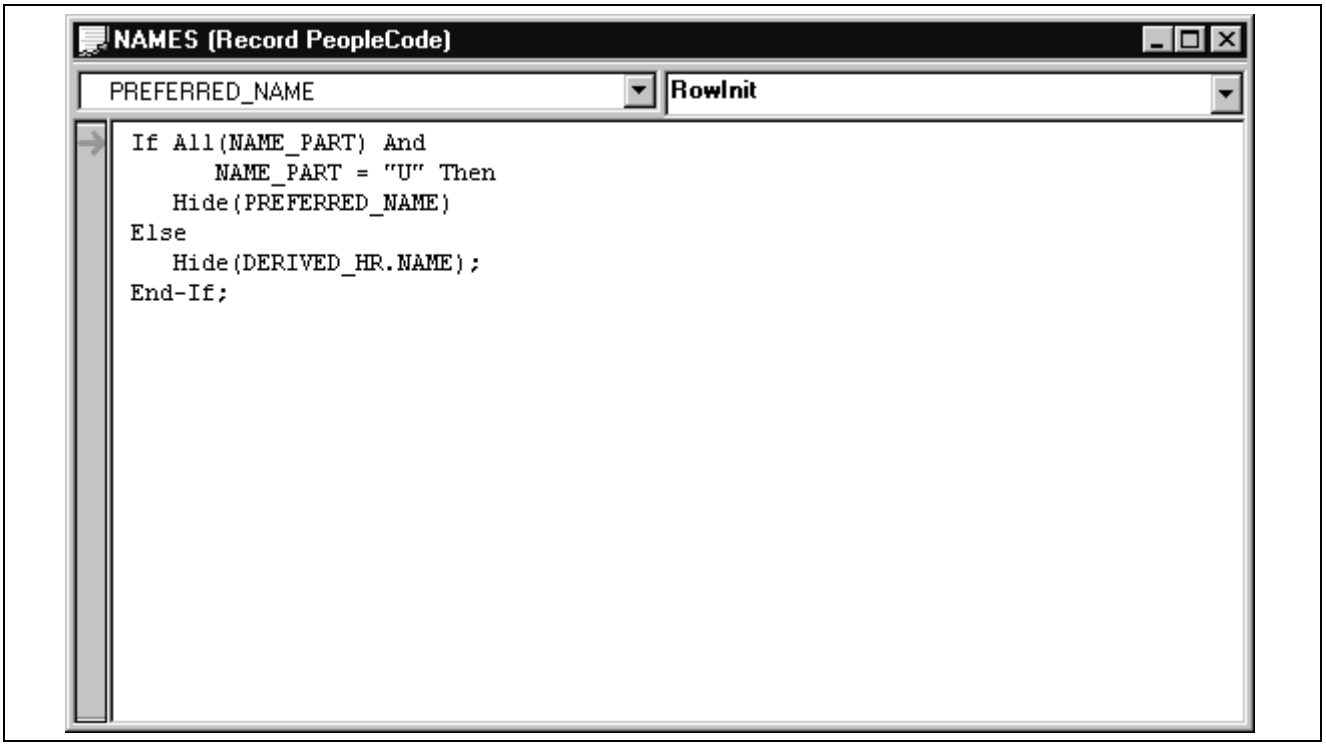
Understanding the PeopleCode Debugger Features

This section discusses:

- Visible current line of execution.
- Visible breakpoints.
- Hover inspect.
- Single debugger.
- Variable panes.
- General debugging tips.

Visible Current Line of Execution

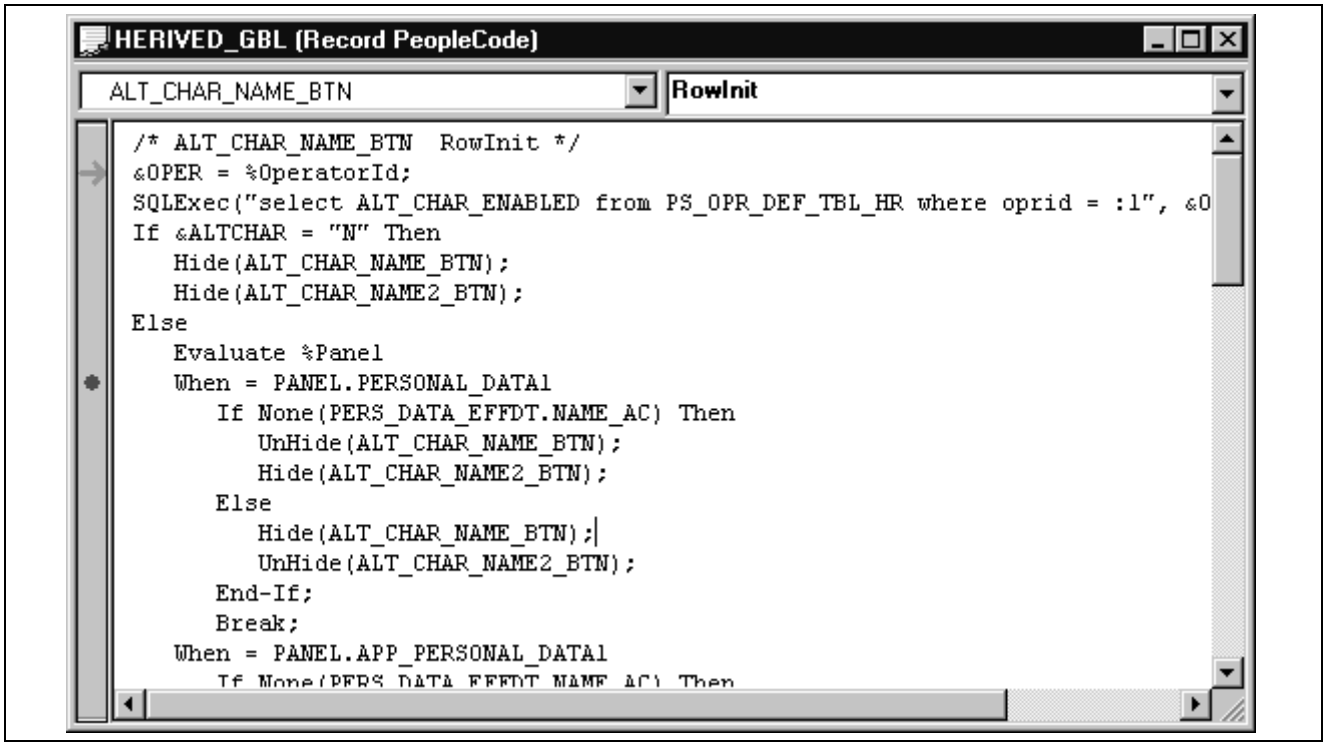
The current line indicator (green arrow displayed in left-hand gutter) is shown in the following illustration:



PeopleCode debugger with current line of execution

Visible Breakpoints

The PeopleCode debugger supports visual indicators that signify breakpoint locations. In the following example, the current line indicator (green arrow) is shown at the first line, and the breakpoint (red dot displayed in left-hand gutter) is on line 8:



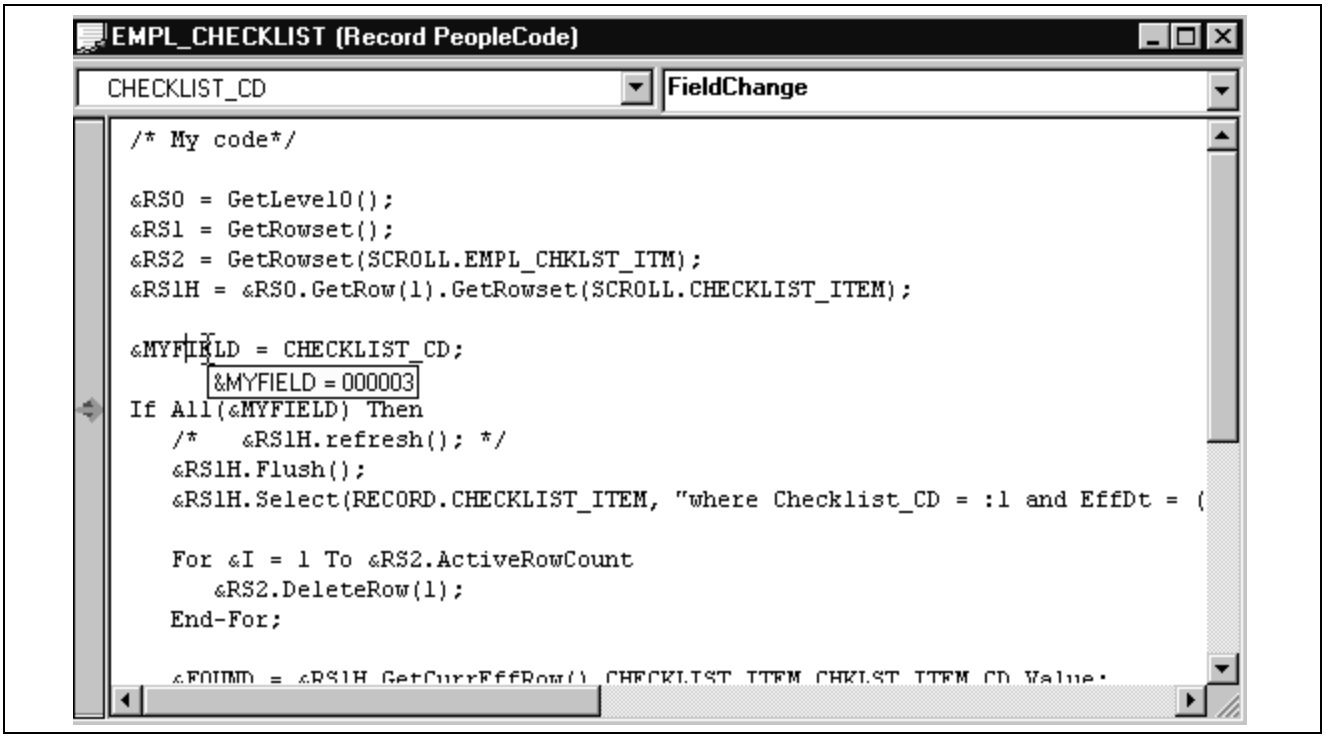
PeopleCode debugger with breakpoint and current line of execution

All breakpoints are saved when Exit Debug Mode is selected.

Note. You cannot set breakpoints on function declarations, variable declarations, or in comments.

Hover Inspect

If the program is already running, you can see the actual values for the variables by holding the cursor over them. The current value is displayed in a pop-up window, as shown in the following example:



PeopleCode debugger with hover inspect

Hover inspect is implemented only for simple variables and fields.

Hover inspect is not implemented for object expressions (for example, rowset assignments and array assignments).

Single Debugger

Each PeopleSoft session you run on a machine can have its own debugging session. However, there can only be one instance of the PeopleCode debugger per session. If more than one instance of PeopleSoft Application Designer is running for a session, only one may be the active debugger at a given time.

From within a running instance of PeopleSoft Application Designer, any component in the same session is also placed into debug mode.

After the session is in debug mode, any component that is started and that belongs to that session automatically goes into debug mode.

Similarly, Application Engine PeopleCode, component interface PeopleCode, and message subscription PeopleCode can be debugged.

After you exit debug mode by selecting Debug, Exit Debug Mode or by exiting PeopleSoft Application Designer, all components in that session go out of debug mode.

If you exit a component, debugging continues with any remaining open and running components.

If more than one PeopleSoft Application Designer session is running, the Application Designer session that is used as a debugger is the first one to be started.

If you're in debug mode, a PeopleCode Editor window is opened for every item (for example, record, component, or page) that has PeopleCode in it when that PeopleCode is executed. If a component has more than one event with a PeopleCode program, only one window is opened per item.

For example, if you have a record that has PeopleCode in both the SearchSave and RowInit events, only one PeopleCode Editor window is opened: first it contains the SearchSave PeopleCode program, then the RowInit program. If you have PeopleCode in the RowInit event for two different records that are part of the same component, two PeopleCode Editor windows are opened, one for each RowInit PeopleCode program.

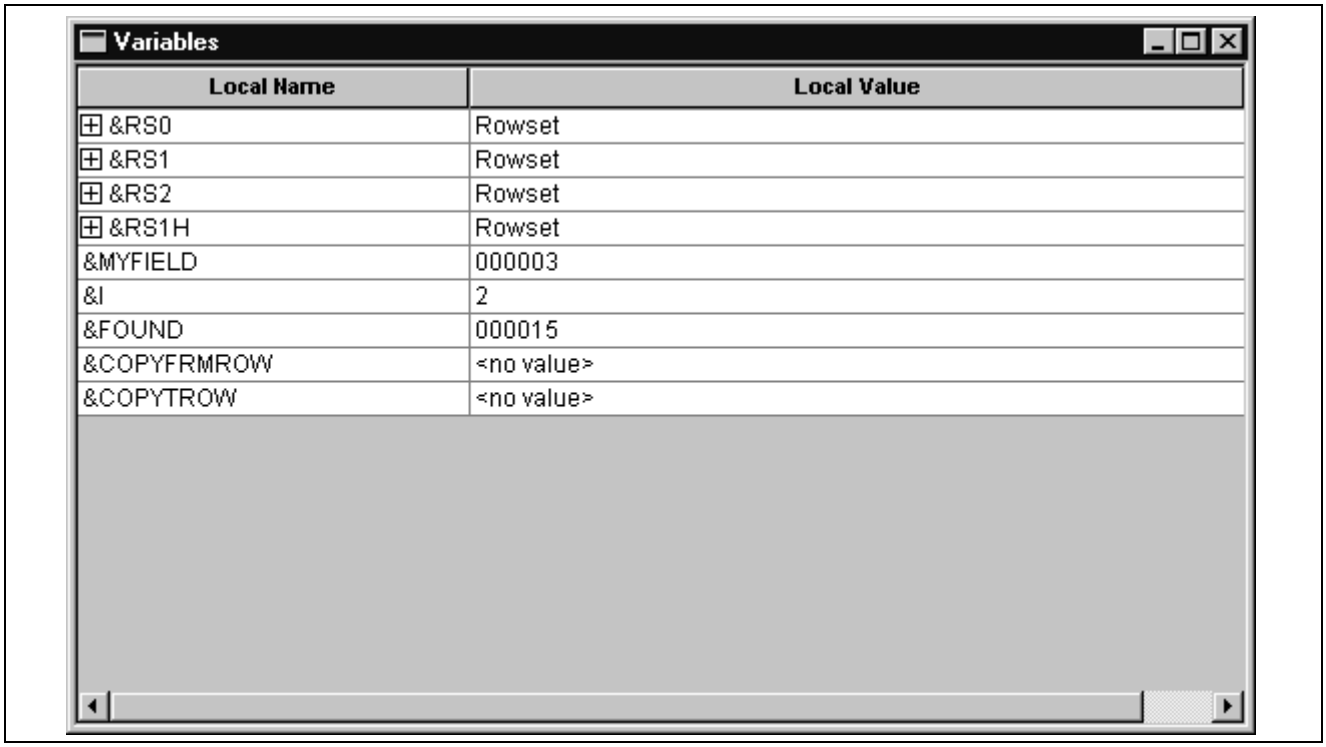
Variables Panes

There are four types of variables panes:

- Local
- Global
- Component
- Parameter

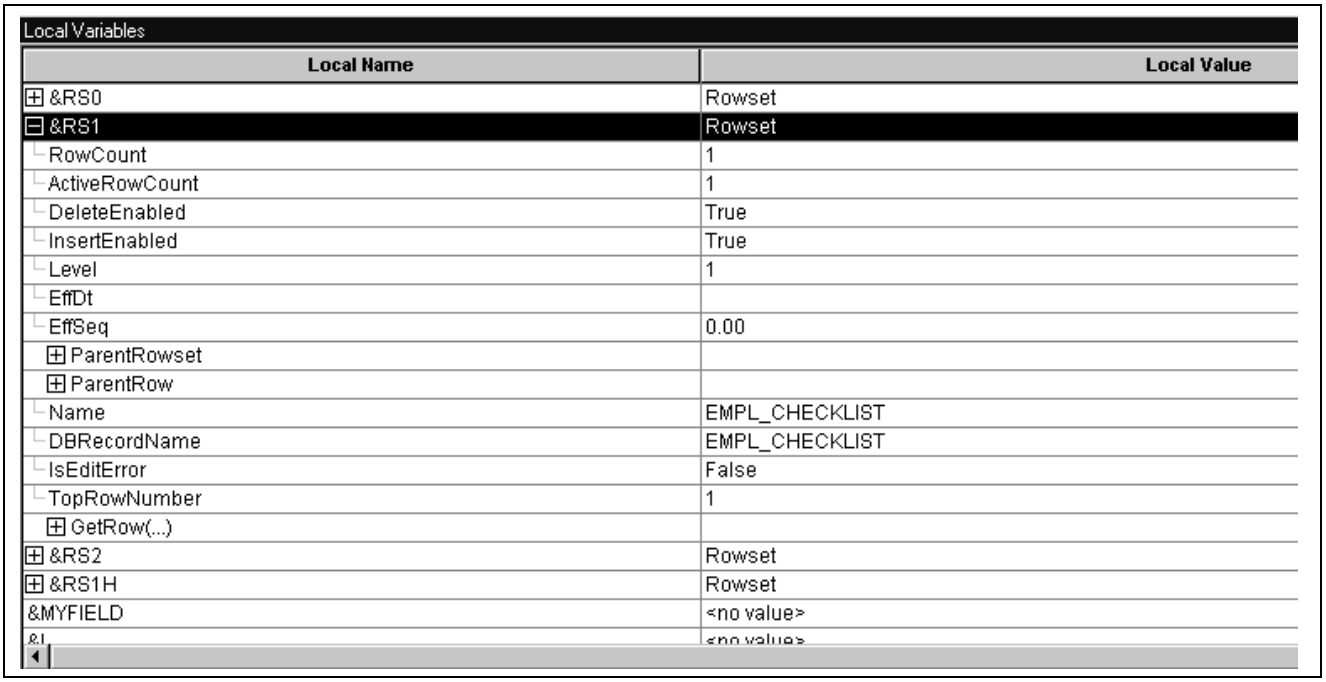
The Local, Global, and Component variable panes show local, global, and component variables, respectively. The Parameter variable pane shows the value of parameters passed in function declarations.

From the variables pane, you can check the value of the variables you have in the program. These values are updated as the code runs. The following illustration shows the variables pane:



Local Variables pane

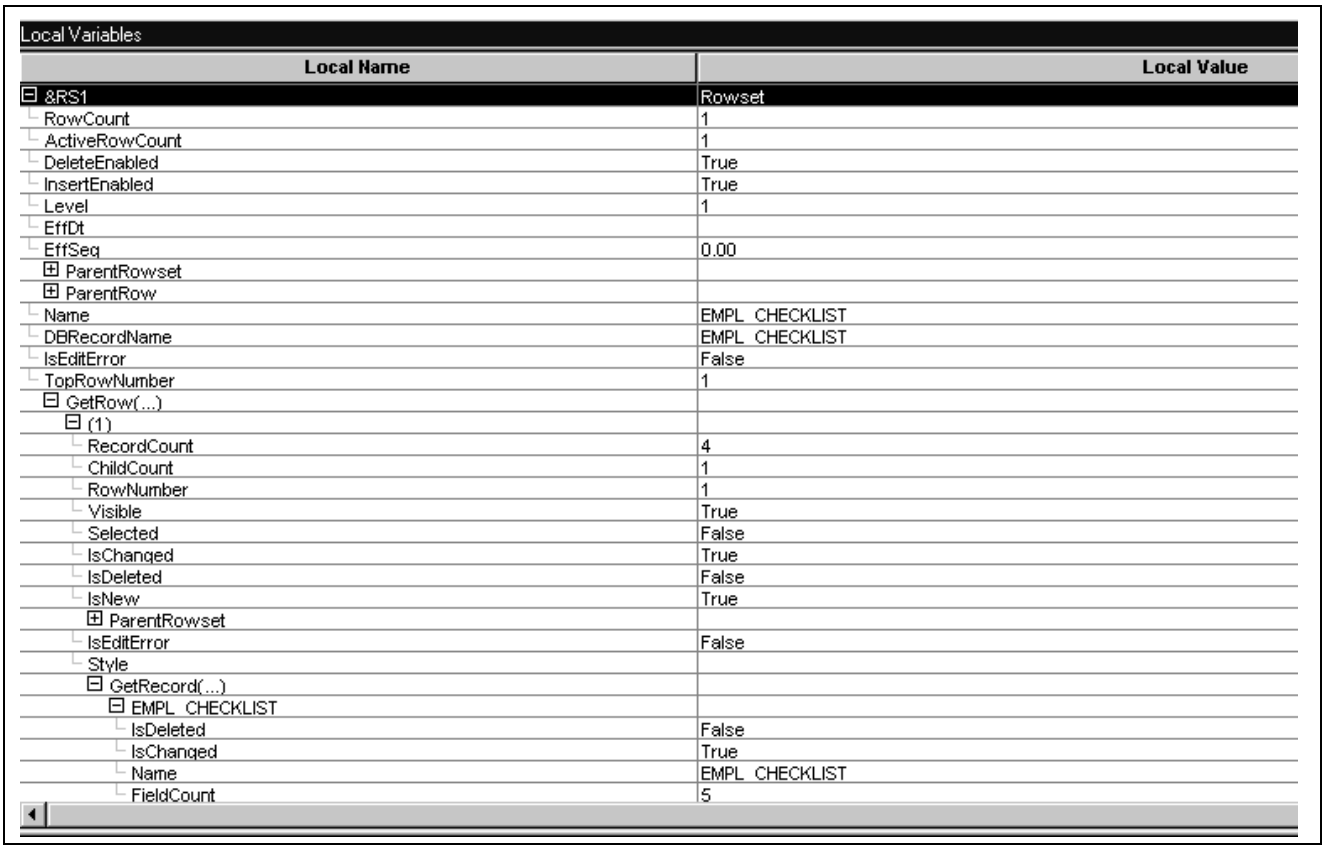
In addition, you can expand any of the objects to see its properties by clicking the plus sign next to the variable name. In the following example, a level one rowset has been expanded. You can see the properties that are part of the rowset (such as ActiveRowCount or DBRecordName).


Local Variables

Local Name	Local Value
⊞ &RS0	Rowset
⊟ &RS1	Rowset
RowCount	1
ActiveRowCount	1
DeleteEnabled	True
InsertEnabled	True
Level	1
EffDt	
EffSeq	0.00
⊞ ParentRowset	
⊞ ParentRow	
Name	EMPL_CHECKLIST
DBRecordName	EMPL_CHECKLIST
IsEditError	False
TopRowNumber	1
⊞ GetRow(...)	
⊞ &RS2	Rowset
⊞ &RS1H	Rowset
&MYFIELD	<no value>
&I	<no value>

Local Variables pane with rowset object expanded

In addition, some objects contain other objects: a rowset contains rows, rows contain records or child rowsets, and records contain fields. You can expand these secondary objects also, to see their properties. In the following example, the first row of a rowset has been expanded, as has the EMPL_CHECKLIST record:


Local Variables

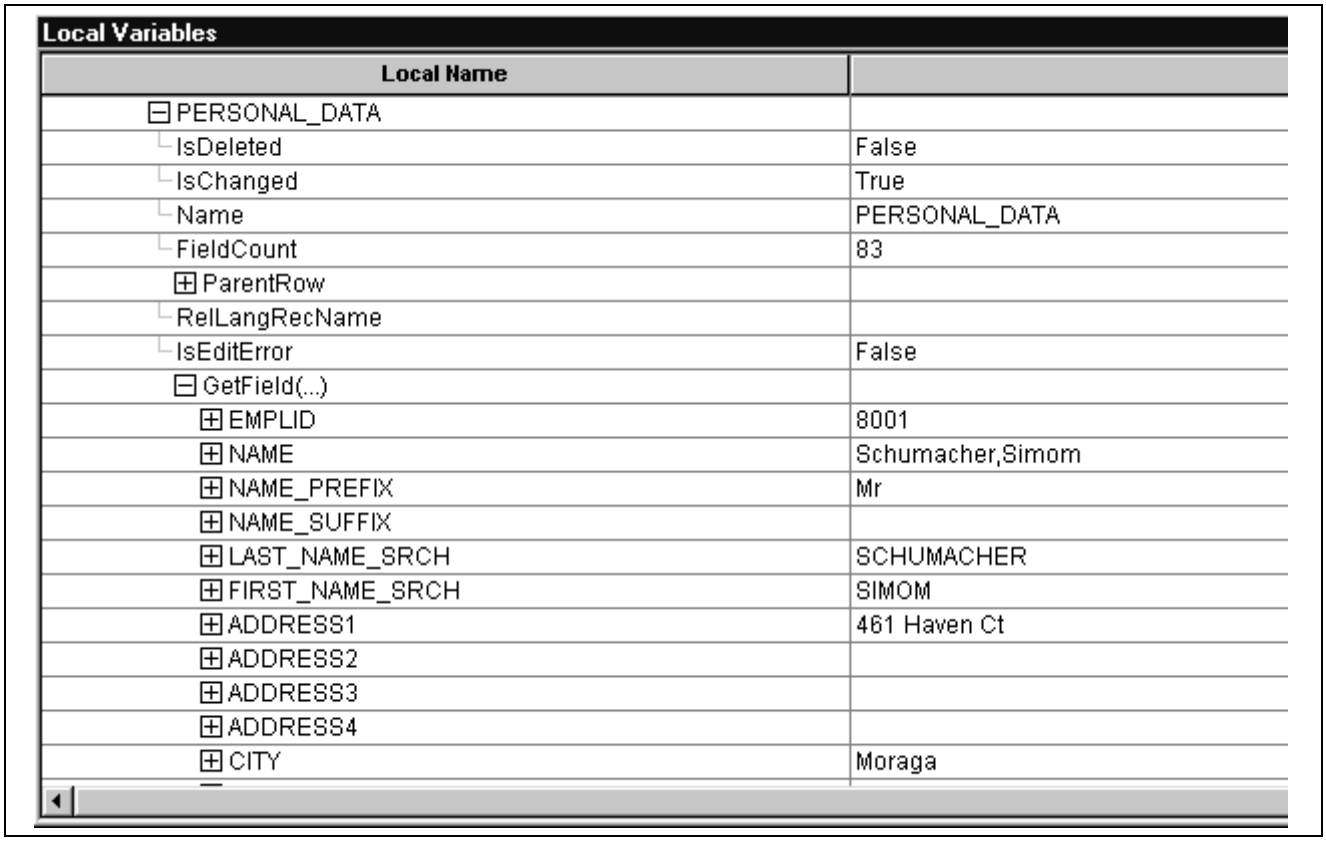
Local Name	Local Value
⊟ &RS1	Rowset
RowCount	1
ActiveRowCount	1
DeleteEnabled	True
InsertEnabled	True
Level	1
EffDt	
EffSeq	0.00
⊞ ParentRowset	
⊞ ParentRow	
Name	EMPL_CHECKLIST
DBRecordName	EMPL_CHECKLIST
IsEditError	False
TopRowNumber	1
⊟ GetRow(...)	
⊟ (1)	
RecordCount	4
ChildCount	1
RowNumber	1
Visible	True
Selected	False
IsChanged	True
IsDeleted	False
IsNew	True
⊞ ParentRowset	
IsEditError	False
Style	
⊟ GetRecord(...)	
⊟ EMPL_CHECKLIST	
IsDeleted	False
IsChanged	True
Name	EMPL_CHECKLIST
FieldCount	5

Variable pane with rowset, row, and record expanded (shown with condensed font)

Field Values

When you view a field object in the debugger, the value of the field is listed under the Value column. Therefore, you do not have to navigate to the Value property to see the value of a field.

The following example shows the PERSONAL_DATA record and the values of the fields:



PERSONAL_DATA record field values

In addition, the only fields that are displayed in the debugger are the fields that are actually in the Component Buffer. For example, suppose you have a derived work record, but don't access all the fields in the work record. Only the fields that you access, and that are in the Component Buffer, are actually displayed in the debugger.

See Also

Chapter 4, "Referencing Data in the Component Buffer," Record Fields and the Component Buffer, page 43

General Debugging Tips

The following are general tips for debugging your application:

- If you're having problems determining if the correct data is being loaded into the component buffers, use the View Component Buffers view window to see all the values currently in the component buffer.

  This is equivalent to putting a GetLevel0 function at the start of a program.

  Use the &LEVEL0 variable to navigate through all the levels of the rowset object, see the row, records, fields, and so on. This shows you everything that has been loaded into the component buffers for that component.

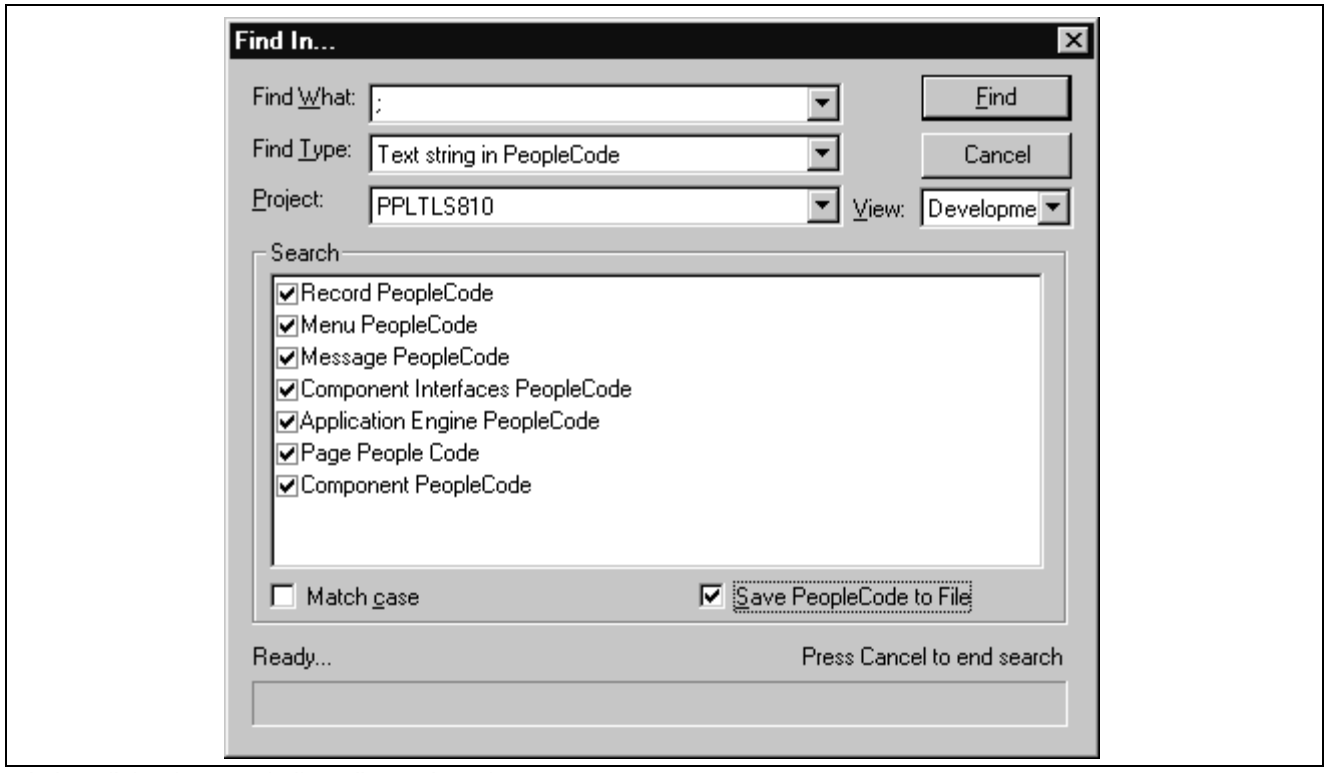
- While at a breakpoint, if you lose track of the window, or the location within the window, that is displaying the green execution location arrow, you can use the Execution Location Properties menu item's ViewCode button to find your current execution location again.
- Objects remain expanded in the variable windows as you move through PeopleCode.

  This enables quick inspection of the state of an object as you step through the PeopleCode. However, there is a performance cost for using this feature. If you are finished examining an object, you may want to collapse it to improve the response speed.
- If a database transaction has been started (either for you by PeopleTools, or by you in PeopleCode) other users of that database are blocked from accessing that database until the transaction is complete.

  If you are stepping through PeopleCode while this transaction is open, you could potentially block other users for an extended period of time. You may want to use a private database for debugging to avoid blocking other users.
- To create a file that contains all the PeopleCode for a project (or database), use the Find In feature and search for `;`.

  Be sure to select *Save PeopleCode to File.*

The following illustration shows the Find In dialog box:



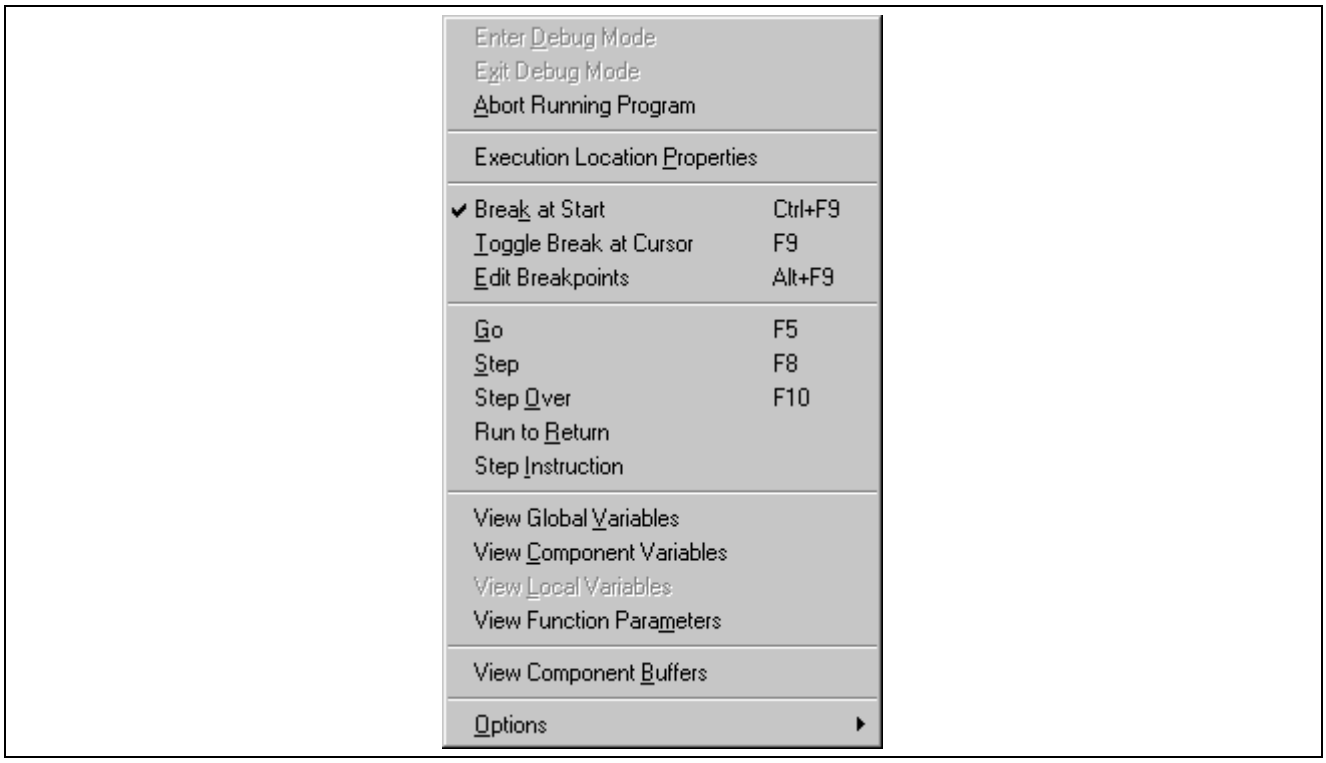
Find In dialog box for finding all PeopleCode

DoModal Considerations

If you have set the PeopleCode debugger to break at start, and you're using the DoModal PeopleCode function, the DoModal window may appear behind the PeopleCode debugger window. This makes it appear as though the debugger has stopped, when it actually has not. Be sure to check that other windows have not opened while you're debugging the code.

PeopleCode Debugger Options

After the debugger is running, you can use the Debug menu to select other options:

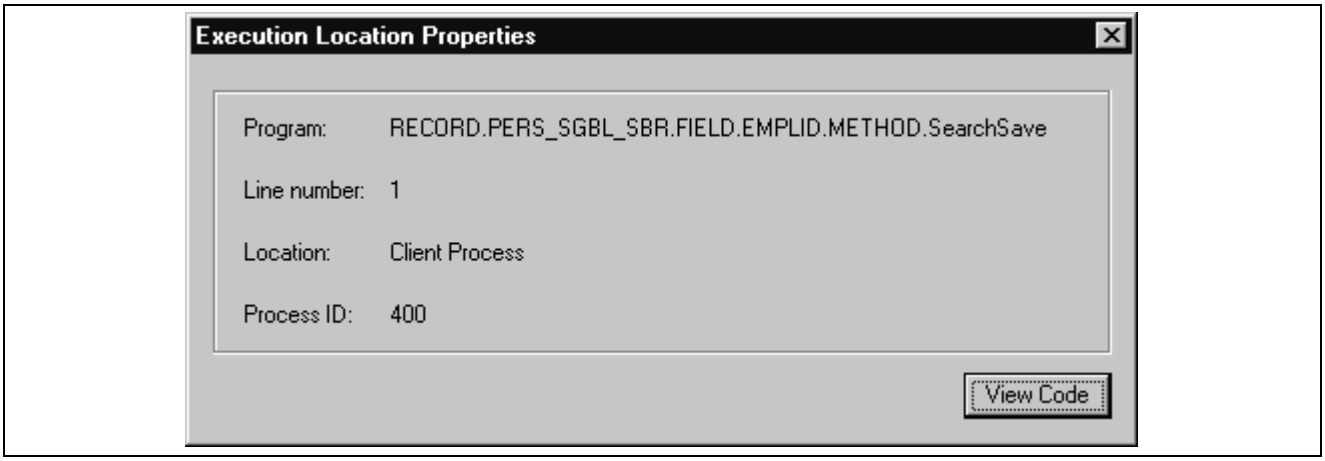


PeopleCode debugger options

Exit Debug Mode Exits debug mode. When you exit debug mode, all breakpoints are automatically saved. If you close PeopleSoft Application Designer, you automatically exit debug mode.

Abort Running Program Stops running the PeopleCode program that is currently running.

Execution Location Properties Displays the location of the running code in a dialog box. This includes the record name, field name, event name, and line number of the code. It also indicates if the code is executing on the client or server. You can also view the exact code by clicking View Code.

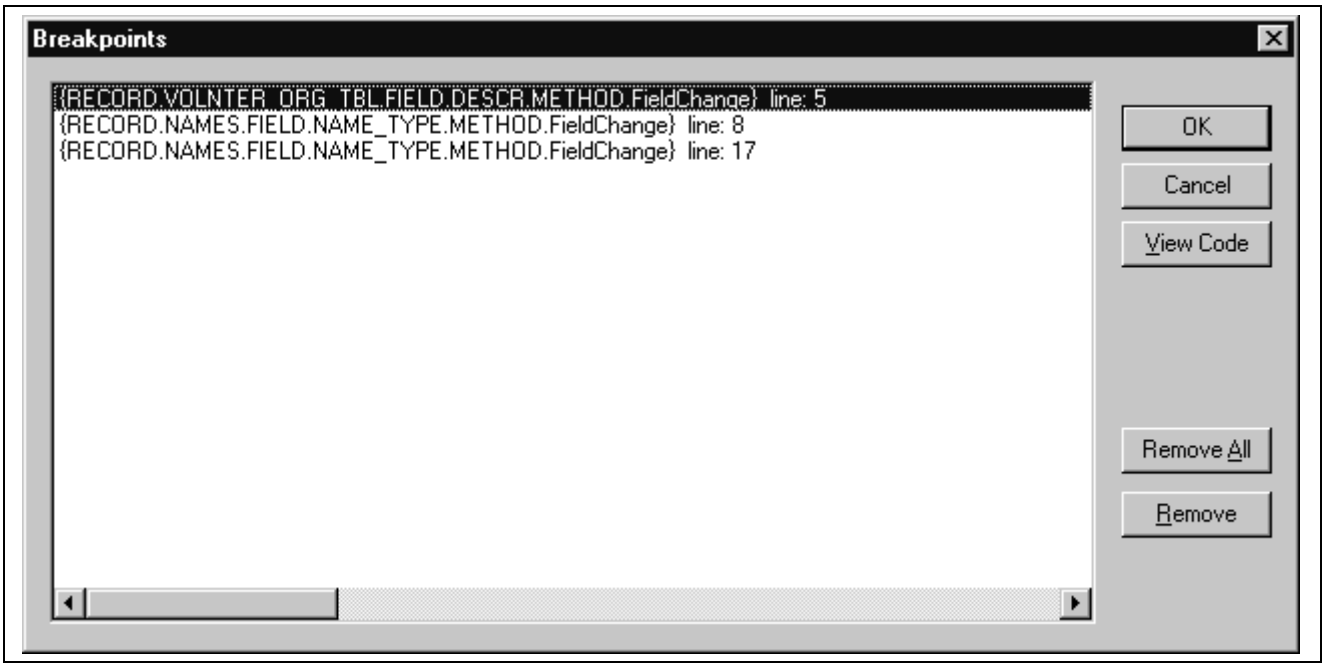


Execution Location Properties dialog box

Break at Start Pauses execution of the component on the first line of every PeopleCode program that executes in the component. If you've started a component with Break at Start selected, and then start a second component, the PeopleCode associated with the second component is stopped at the first line of the first PeopleCode program as well, as part of the same debugging session.

Toggle Break at Cursor Removes the breakpoint if the line the cursor is currently on has a breakpoint. Adds a breakpoint if the line the cursor is currently on does not have a breakpoint.

Edit Breakpoints Opens a dialog box that displays the lines that have breakpoints. From this dialog box, you can display the code that contains the breakpoint by clicking View Code. You can also remove one or all breakpoints.



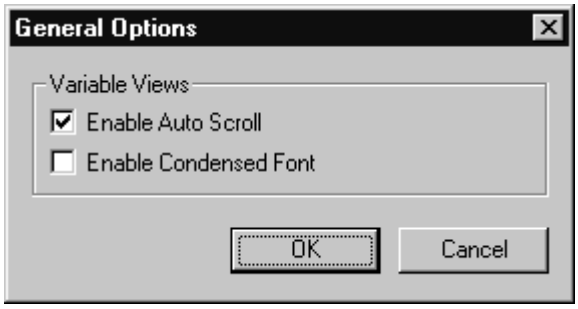
Breakpoints dialog box

Go Continues processing until the next breakpoint. If Break At Start is enabled, processing pauses at the next PeopleCode program.

Step	Executes the current line of the PeopleCode program, stepping into functions.
Step Over	Steps through each line of the PeopleCode program, one line at a time, but steps over the functions; the functions are executed, but not stepped into.
Run to Return	Processes past the return of the current function, then pauses.
Step Instruction	Processes low-level pseudo-machine code instructions internal to PeopleCode. This option is used in conjunction with Log Options.
View Global Variables	Opens a separate window for watching global variables.
View Component Variables	Opens a separate window for watching component variables.
View Local Variables	Opens a separate window for watching local variables.
View Function Parameters	Opens a separate window for watching user-specified parameters in function calls.
View Component Buffers	Opens a separate window for viewing the current component buffers. This is equivalent to getting a level zero rowset for the component.

Note. These five windows are continuously updated as the program executes.

Options	Enables you to select between opening up a dialog box for general options or for specifying log options.



General Options dialog box

The General Options dialog box enables you to specify conditions of the view windows. The default is for both of these options to be selected.

Enable Auto Scroll	If you select this check box, and you click a plus symbol next to a variable name in a view window, the variable you clicked scrolls to the top of the window.
Enable Condensed Font	Select to display all view windows with a smaller font.

Additional Features

Break at Termination	After you are in debug mode, generally, any PeopleCode program in the session that terminates abnormally first breaks in the debugger. In addition, the error message is displayed in the PeopleCode log in the bottom window of PeopleSoft Application Designer.

See Also

Chapter 16, “Debugging Your Application,” Setting PeopleCode Debugger Log Options, page 264

Setting Up the Debugging Environment

You can use the PeopleCode debugger for two-tier and three-tier debugging. The database and application can reside on remote servers; they do not need to reside on the local machine.

Two-tier debugging works out of the box. Setting up three-tier debugging requires you to make a few modifications in PSADMIN (PSAPPSRV.CFG) to enable debugging.

You can connect to a Microsoft Windows NT server domain that is not on your local machine. You don't have to configure a local domain to do this. You also don't have to have PeopleTools software installed locally for three-tier debugging.

See Also

Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration, "Configuring Trace and Debug Settings," Setting Up the PeopleCode Debugger

Debugging Subscription PeopleCode

To debug subscription PeopleCode:

1. Start PeopleSoft Application Designer on the server machine that will be processing the subscription.

   You must use the same user ID and database as the machine to which its application servers have been configured.

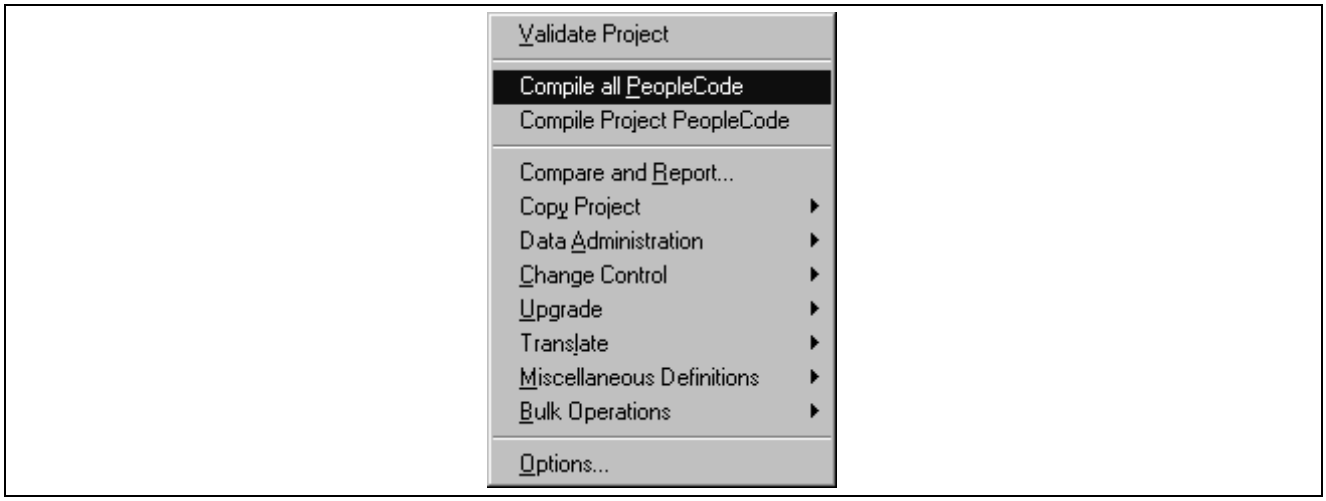
2. Start debug mode.

   When the subscription server executes the subscription PeopleCode, the debugger starts.

Note. You cannot use the PeopleCode debugger for OnRequest for a local sync message (SyncRequest or SyncRequestXmlDoc). A possible workaround for local sync PeopleCode debugging would be to actually boot up two application servers and point the local node on the gateway to the application server that is not making the SyncRequest (SyncRequestXmlDoc) call.

Compiling All PeopleCode Programs at Once

In addition to checking individual programs, you can compile all PeopleCode programs either in a database or in a project to check for errors. This option opens and compiles every PeopleCode program. This process can be run on an as-needed basis to check for corruption in your programs. Run this option after an upgrade to verify that all the programs were upgraded correctly. You run this option from the Tools menu, as shown in the following illustration:



Tools menu - Compile All PeopleCode option

To compile all PeopleCode programs:

1. Open PeopleSoft Application Designer while accessed to the database that contains the PeopleCode that you want to check.
2. Select the compile option to use.

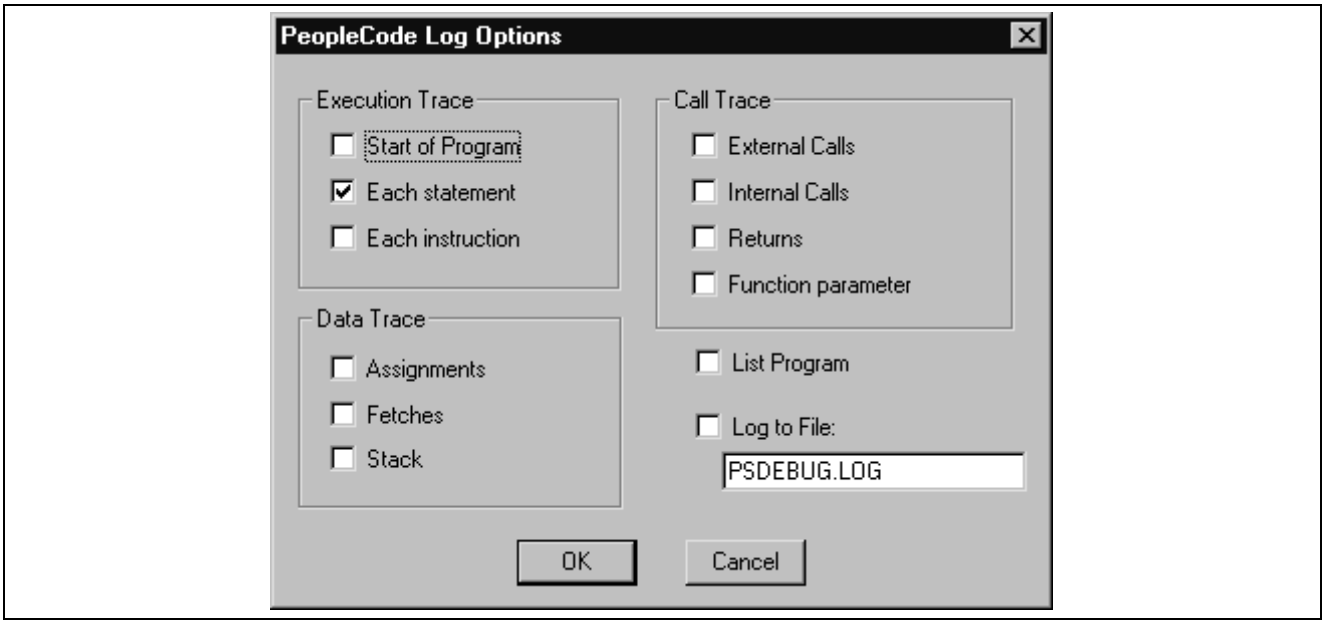
   Select Tools, Compile All PeopleCode or Tools, Compile Project PeopleCode.
3. Click Compile in the Compile All PeopleCode dialog box.

   Errors appear in the PeopleCode log display window.

Note. If you've specified a log file in the debugger log options, all errors are written to the log file as well.

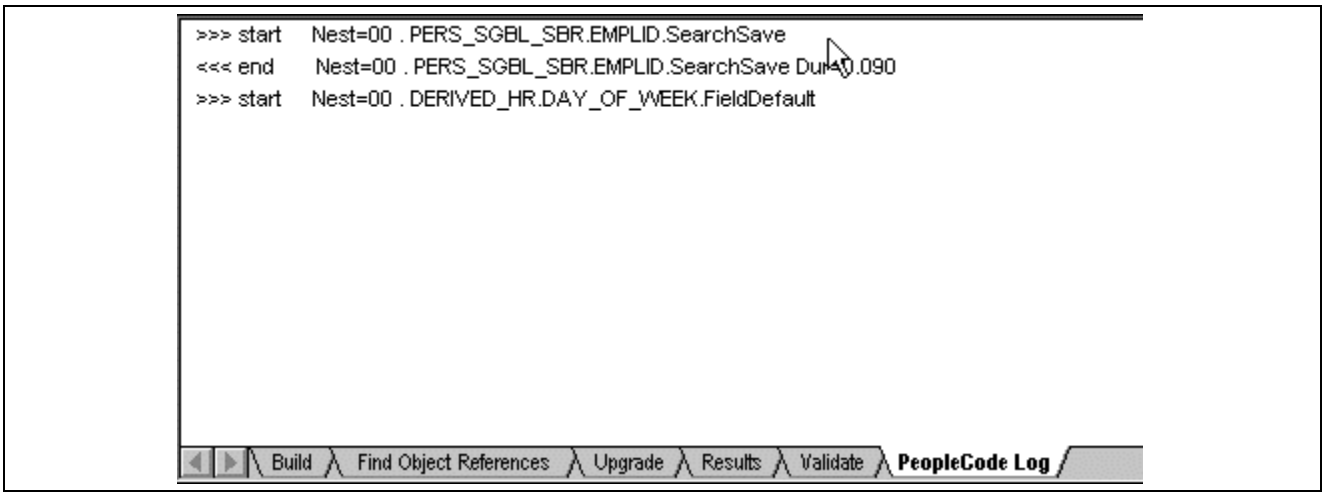
Setting PeopleCode Debugger Log Options

Use the PeopleCode debugger to view PeopleCode that is executed while you're stepping through your application. Select Debug, Log Options to access the PeopleCode Log Options dialog box.



PeopleCode Log Options dialog box

All log information appears in the PeopleCode log window, at the bottom of PeopleSoft Application Designer.



PeopleCode log window

You can also record what you see in a log file. You can tailor the log results to record a variety of online information.

If you exit debug mode, but do not close PeopleSoft Application Designer, all the log options that you specified are still there when you start debug mode again.

When you close the PeopleSoft Application Designer, all log options are clear. The next time you enter debugging mode, you must reselect debug log options.

See Chapter 16, "Debugging Your Application," Interpreting the PeopleCode Debugger Log File, page 267.

All the options available in the Log Options dialog box are also available in PeopleSoft Configuration Manager, on the Trace tab, in the PeopleCode Trace section.

Execution Trace Options

Execution trace is set to trace each PeopleCode statement. You can also trace the start of each program or each program instruction.

Data Trace Options

The data trace options are:

Option	Description
Assignments	Records each assignment made to a field.
Fetches	Records the field values retrieved from a PeopleCode fetch.
Stack	Indicates the contents of the internal machine stack. Typically, only PeopleSoft staff developing PeopleCode language enhancements use this option.

Call Trace Options

The call trace options, described in the following table, enable you to record the values of external calls, internal calls, returns, and function parameters.

Option	Description
External calls	Traces each call to external (PeopleCode) functions.
Internal calls	Records each call to internal subroutines.
Returns	Logs the occurrence of program returns.
Function parameters	Logs the value of individual PeopleCode function parameters.

Log To File

When you select this option, you must specify the name of a file, or you receive an error and logging to file is disabled.

If you do not specify a directory location, the file is placed in the same directory from which you're running PeopleTools.

If you specify the name of an existing file, a warning message appears, asking you whether to overwrite the file. At this point, you must go back into the Log Options dialog box and specify a different file name, or else the log file is overwritten.

If you do not exit PeopleSoft Application Designer before running a different application, each trace is appended to the specified log file.

See Also

Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration, "Using PeopleSoft Configuration Manager," Specifying Trace Settings

Interpreting the PeopleCode Debugger Log File

You can produce a trace log using any of the following methods:

- Using the Log File option in the PeopleCode debugger.
- With the PeopleSoft Configuration Manager Trace tab.
- Using the SetTracePC and SetTraceSQL built-in functions.
- With PeopleTools utilities (included for backward compatibility purposes only and should not be used).

All trace files except those produced using the Log File option contain timing information, such as when each line started processing and how long it took to execute.

The Log File option writes to a file that you specify. The log file produced by the other options is specified by the PeopleTools Trace File option in PeopleSoft Configuration Manager. All of these options write to the same file.

Trace files are also produced by PeopleSoft Application Engine. These logs may contain more information.

This section discusses:

- Log file contents.
- Other items in the log file.

See Also

Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," Using Debug Utilities

Log File Contents

The log file contains useful information for debugging PeopleCode.

You can view the log using any editor that displays ASCII text, such as Notepad. the log file has the following components.

Line Count	Specifies a line number in the file.
Internal Information	Contains reference numbers used for internal tracing. You can ignore this information.
Instruction Location	Address of an instruction processed in the program. You can follow programs and functions using this number.
Operation Code	Indicates the operation performed by the program.
Operation Operands	Contains information specific to each operation. The following table lists the possible operations and the operands that appear for the list and trace options.

Other Items in the Log File

The following table describes other items that can appear in a debugging trace:

Trace Item	Description
Store Field:record name.field name Value=xx	Issued when the assignments trace option is selected. It contains the record and field names and the value that is stored.
Fetch Field:recordname.fieldname Value=xx	Issued when the Fetch Field option is selected. It contains the record and field name and the value that is retrieved.
Fetch Field:recordname.fieldname Contains Null Value	Issued when the Fetch Field option is selected and the selected record.field contains a null value.
Fetch Field:recordname.fieldname Does Not Exist	Issued when the Fetch Fields option is selected and when the field is not found.
Branch Taken	Displayed after a branch test when the branch is taken.
Field Not Found, Statement Skipped	Displayed whenever a referenced field was not found error causes the PeopleCode processor to skip to the next statement.
vvvvvv PeopleCode Program Listing	Issued when the List Program option is selected. It marks the beginning of a PeopleCode program listing.
^^^^^ PeopleCode Program Listing End	Issued when the List Program option is selected. It marks the end of a PeopleCode program listing.
Error Return -> NNN	Issued when a fatal error condition terminates the PeopleCode program.

Using Application Logging

Application logging enables you to do error logging using an independent application log fence mechanism, as well as being able to write to the PeopleTools log using the WriteToLog built-in function.

Note. This is an application log fence, and is distinct from the PeopleTool's LogFence setting.

In PeopleTools, a log fence is a type of "barrier." Application error messages are only written to the PeopleTools log if the log fence setting that the messages are written to the log with (using WriteToLog) is less than or equal to the current application log fence setting (AppLogFence) in the application server configuration file (PSAPPSRV.CFG .)

For example if the AppLogFence setting is 2, only messages written using the WriteToLog function with a log fence value less than or equal to 2 will be written. This allows you to have application logging code written in your application that will only be in effect if the log fence setting permits.

The application log fence setting is available through the system variable %ApplicationLogFence.

Apart from the obvious use of allowing the application to write to the Tool's log file, this mechanism is also an aid in debugging. For example, you could interleave PeopleCode, SQL, and application level tracing in the same logfile to easily correlate application and PeopleTools actions.

See Also

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

Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Language Reference, "System Variables," %ApplicationLogFence

Setting the Application Log Fence in the Configuration File

The Application log fence defaults to %ApplicationLogFence_Level1 (3). If you want to use this setting, you need to place it in the application server configuration file (PSAPPSRV.CFG.) The setting is dynamic change enabled (that is, if it's value is changed in the file then the new value will be used). As the example below illustrates, the AppLogFence seting must be in the PSTOOLS section. If you add this setting, your configuration file can look like this:

```
[PSTOOLS]
;=================================================================
; General settings for PSTOOLS
;=================================================================
AppLogFence=1
```

See Also

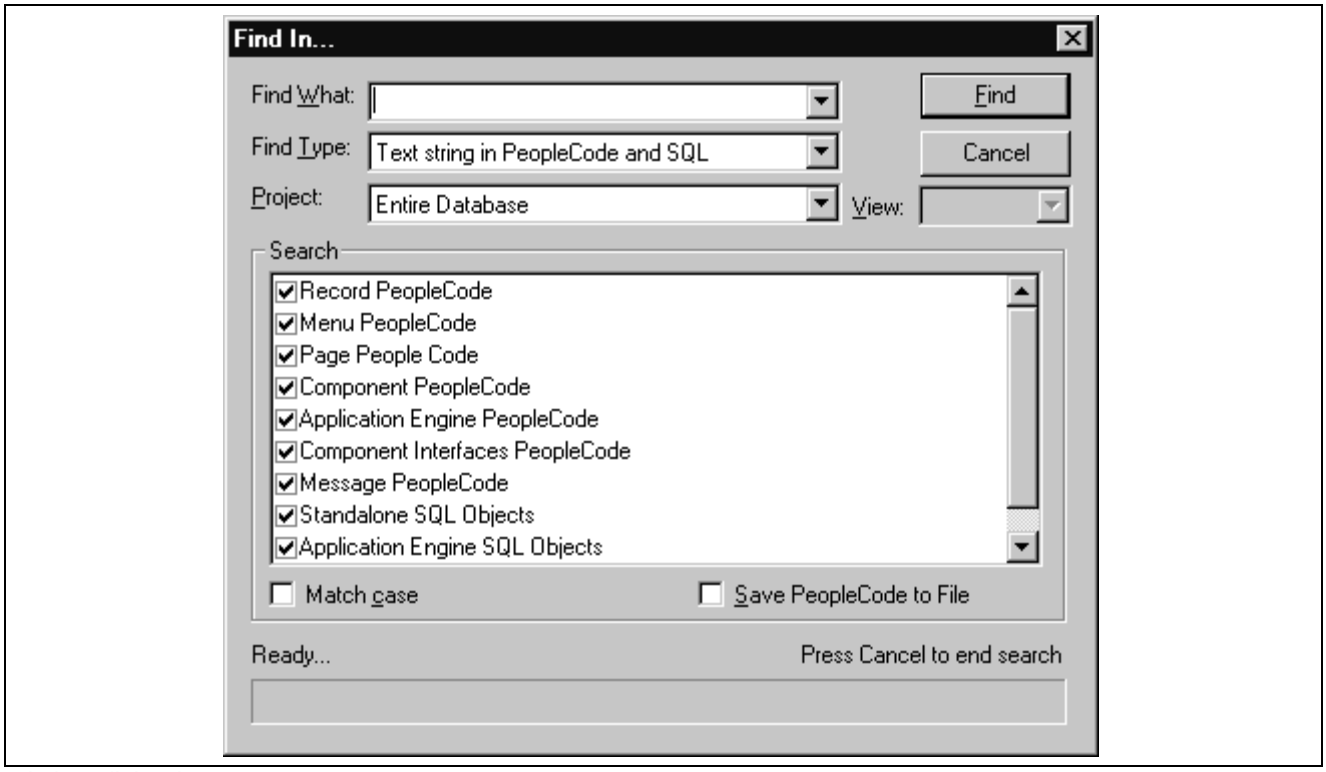
Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration, "Setting Application Server Domain Parameters," PSAPPSRV Options

Using the Find In Feature

Use the Find In feature of PeopleSoft Application Designer to search for strings, either in PeopleCode programs or in SQL definitions. This feature searches:

- All PeopleCode programs and all SQL statements.
- Only PeopleCode programs.
- Only SQL statements.
- Only HTML definitions.
- SQL injection in PeopleCode.

The following illustration shows the Find In dialog box:



Find In dialog box

You can further refine your search by specific project. If you're searching PeopleCode programs and SQL statements, you can specify if you want record PeopleCode, page PeopleCode, menu PeopleCode, and so on.

All output from the search is placed in an output window. You can save these results to a file, copy them, clear them, or print them.

From the output window, you can immediately open any of the PeopleCode programs, SQL statements listed, or HTML.

Also, from the output window, you can insert selected records into a project. Then, if you need to search those records again, you can search by project.

Note. To create a file that contains all the PeopleCode for a project (or database) you can use the Find In feature and search for `;`. Be sure to select *Save PeopleCode to File*.

To find a text string in PeopleCode or SQL:

1. In PeopleSoft Application Designer, select Edit, Find in.

   This displays the Find In dialog box.

2. Type the string that you want to find in the Find What edit box.

   If you want only those items that match the case of what you entered, select the Match Case check box at the bottom of the dialog box.

3. Specify with the Find Type edit box whether you are searching in PeopleCode and SQL, just PeopleCode, just SQL, or HTML.

4. Select the project to search.

   You can search the entire database or any existing project.

5. (Optional) Select the view to search.

   If you decide to not search the entire database, you can specify if you want to search the Development view or the Upgrade view. The default is the Development view.

6. Select the items to search.

   You can search all items that contain either PeopleCode or SQL, or just a subset of items.

   Note. The Search check box list for HTML is empty. The search is conducted against all HTML definitions.

7. (Optional) Save the search results to a file.

   You can save the results of a PeopleCode search to a text file, which you can view or print using a text editor or word processor. The text file contains the entire PeopleCode program that contained the string.

   To save your results to a file, select the Save PeopleCode to File check box at the bottom of the dialog box. The results are saved to the file, and appear in the PeopleSoft Application Designer Find In output window.

8. Click the Find button to start the search.

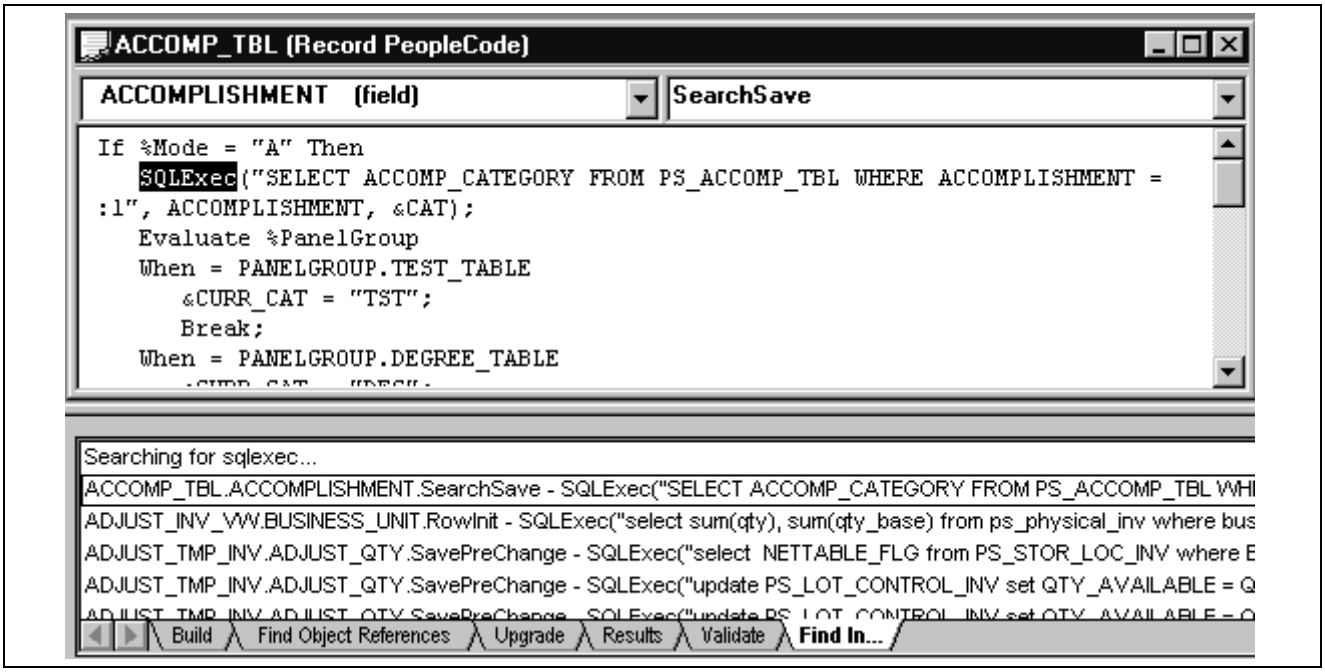
   As the Find In feature searches the database, it displays a counter at the bottom of the Find In dialog box indicating the number of PeopleCode programs searched.

   You can click the Cancel button to stop the process.

9. Check the Find in Output window for results.

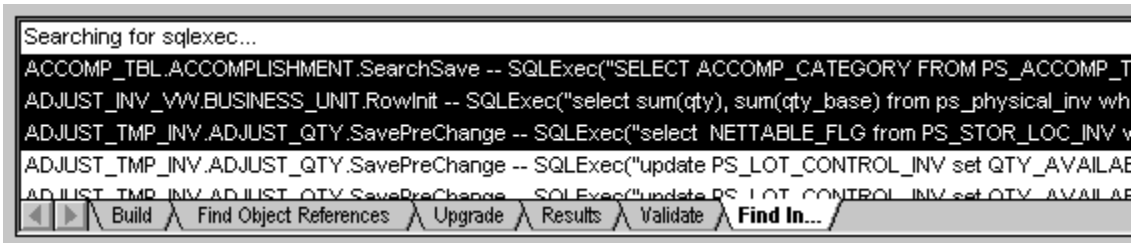
   The results of the search appear in the Find In tab of the output window. Each line shows where the string was found. You can open any of the programs listed by double-clicking a line in the output window.

The following illustration shows the Find In tab of an output window:



Opening a PeopleCode program from the Find In tab

To save records, you select them in the output window, as shown in the following illustration:



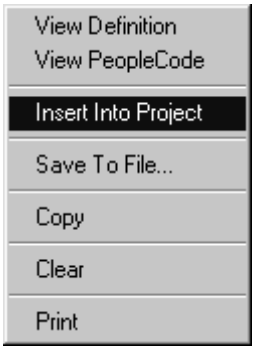
Find In output window with records selected

To save records in a project:

1. Use the Find In feature to search for a string.
2. Press the SHIFT key while selecting the references to save in the output window.
3. Right-click the highlighted records and select Insert Into Project.

   All the selected records are inserted into the current open project.
4. Save your project.

The following illustration shows the Insert Into Project option:



Find In pop-up menu

The next time that you search, you can search only your project (select a project in the Find In dialog box) instead of searching the entire database.

Searching for SQL Injection

SQL injection is a technique that enables users to pass SQL to an application that was not intended by the developer. SQL injection is usually caused by developers who use string-building techniques to generate SQL that is subsequently executed.

Search PeopleCode for SQL injection vulnerabilities.

See Also

Chapter 17, “Improving Your PeopleCode,” Searching PeopleCode for SQL Injection, page 287

Cross-Reference Reports

If a field value changes, and you do not know how it changed, there are two ways of finding all references to a field:

- The Find References option in PeopleSoft Application Designer.
- Cross-reference reports.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer*, "Working With Projects".

PeopleTools is delivered with three PeopleCode cross-reference reports:

- XRFFLPC.

  Reports on all fields in the system referenced by other PeopleCode programs. The report sorts by record names, then field names. XRFFLPC shows the records, fields, and PeopleCode program types that reference each field.

- XRFPCFL.

  Reports on the fields that each program references. It sorts the report by record definition, field name, then PeopleCode type. It shows the records and fields referenced for each program. This report and XRFFLPC complement each other by using converse approaches to reporting the cross references.

- XRFPNPC.

  Reports on pages with PeopleCode. This report shows pages containing fields with PeopleCode attached to them.

You can run these reports using PeopleSoft Query and either view the reports online or print them. You can also download them to a Microsoft Excel spreadsheet. The following example shows an XRFPNPC report:

	Panel Name	Record	Field Name	Prog Type
1	ACL_APPDES_MISC	DERIVED_APPDESM	ACCESS_CD	FieldChange
2	ACL_APPDES_MISC	DERIVED_APPDESM	ACCESS_CD	RowInit
3	ACL_APPDES_MISC	DERIVED_APPDESM	FULL_ACCESS_BTN	FieldChange
4	ACL_APPDES_MISC	DERIVED_APPDESM	ITEM_CHANGED	SavePreChange
5	ACL_APPDES_MISC	DERIVED_APPDESM	NO_ACCESS_BTN	FieldChange
6	ACL_APPDES_MISC	DERIVED_APPDESM	READ_ACCESS_BTN	FieldChange
7	ACL_APPDES_OBJECTS	DERIVED_APPDESO	ACCESS_CD	FieldChange
8	ACL_APPDES_OBJECTS	DERIVED_APPDESO	ACCESS_CD	RowInit
9	ACL_APPDES_OBJECTS	DERIVED_APPDESO	FULL_ACCESS_BTN	FieldChange

XRFPNPCPeopleSoft Query results

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Query, "PeopleSoft Query Preface"

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer, "Running PeopleTools Cross-Reference Reports"

CHAPTER 17

Improving Your PeopleCode

Developer changes can affect how a user interacts with the page. Performance and screen flicker, which occurs whenever the screen refreshes after a server trip, are significant issues.

This chapter discusses how to:

- Reduce server trips.
- Use better coding techniques for performance.
- Write more efficient code.
- Search PeopleCode for SQL injection.

Reducing Trips to the Server

This section discusses how to:

- Count server trips.
- Use deferred mode.
- Hide and disable fields.
- Use the Refresh button.
- Update totals and balances.
- Use warning messages.
- Use the fastest algorithm.

Server trips are bad for performance. Each server trip consumes resources on the application server, slows down the user data entry, and can affect type ahead. Whenever you see an hourglass as you move between fields on a page, it is because the browser is waiting for a server trip to complete.

The larger the component's buffer (based on the number of record definitions accessed, the number of fields in each record, and the number of rows in each grid or scroll area for each record), the longer each round trip to the server, because of the increased server processing. Server trips also create a distracting page flicker as the page is redisplayed after each server trip.

Deferred mode reduces the user's time to complete the transaction and conserves application server resources.

The following user interactions cause a trip to the server. Only the first three items in the list are deferred in deferred processing mode.

- Entering data in fields with FieldEdit or FieldChange PeopleCode.
- Entering data in fields that have prompt table edits.

- Entering data in fields that have related displays.
- Inserting a row in a grid or scroll area.
- Deleting a row from a grid or scroll area.
- Using grid or scroll area controls to move forward or back.
- Accessing another page in the component.
- Selecting an internal tab.
- Expanding or collapsing a collapsible section.
- Clicking a button or link.

Each trip goes through the same process of checking security, unpacking the buffers that store the data being processed, processing the service request, generating the HTML for the page to be redisplayed, packing updated buffers, and storing the buffers on the web server. To maximize online performance, minimize server trips.

Counting Server Trips

Count the trips to the server to quickly identify transactions that have performance issues. PeopleTools can automatically count these trips by reason (such as,adding a row in a grid or FieldChange PeopleCode) and write the output to a log file.

To turn this feature on, run a debug version of PeopleTools and add the following to the [trace] section of the appserv.cfg file:

```
showcounters = 1
```

The output is written to the appsrv.log file.

Using Deferred Mode

Keep components in deferred mode and enable fields for interactive mode only if there is a strong business case.

In order for every field on the component to run in deferred mode, Deferred mode must be selected at the component level, Allow Deferred Processing must be selected for each page in the component, and Allow Deferred Processing must be selected for each field.

PeopleSoft recommends that you continue to code field edits in FieldEdit PeopleCode and field change logic in FieldChange PeopleCode, but set this logic to run in deferred mode. You do not need to move field edits to SaveEdit.

Hiding and Disabling Fields

Avoid using FieldChange PeopleCode to hide, unhide, enable, or disable elements on the same page, unless the element is triggered by a separate button.

Hiding or unhiding objects and enabling or disabling objects should, as a general rule, be coded in either page Activate PeopleCode, or in FieldChange PeopleCode for objects that are on another page in the component.

Perform cross-validation edits to prevent invalid data combinations from being written to the database for fields that previously would have been hidden or unavailable. If unhiding fields that were previously hidden or unavailable results in making the page confusing, consider designing a longer page, so that users can easily associate related fields.

You can hide or unhide objects or set them to display-only in page Activate PeopleCode before the page is initially displayed, based on setup data, configuration options, or personalization settings. Fields can be set to display-only using PeopleCode by setting the DisplayOnly property for the field to True.

You can hide or unhide fields on another page, or set the fields to display-only, based on the value that a user enters in a field on the current page, as long as that component or field is set up to run in deferred processing mode. In some cases, it may make sense to split transactions across pages to achieve progressive disclosure.

Using the Refresh Button

The Refresh button gives users control of their environment. Clicking the Refresh button forces a trip to the server. PeopleTools then redisplays the page in the browser. This allows the user to:

- See related display field values for the data entered so far.
- See current totals or balances that are calculated based on lines entered in a grid or scroll area.
- See any default values based on prior data entered on the page.
- Validate the data that has been entered on the page so far.

When the page is redisplayed, the cursor is positioned in the same field it was when the user pressed the Refresh button.

Updating Totals and Balances

In some pages, totals or balances are displayed based on data entered into a grid or scroll area. This process should work in deferred mode also, showing the totals or balances as of the last trip to the application server.

Continue to keep any accumulation and balancing logic in FieldChange PeopleCode, but run the field in deferred mode. This enables users to click the Refresh button at any time to see the latest totals based on the data entered. Totals and balances in deferred mode are always updated and displayed after any trip to the application server.

Using Warning Messages

In deferred mode, FieldEdit PeopleCode errors and warnings are not displayed when a user moves out of the field, but rather on the next trip to the server. This might not occur until the user enters all the data and clicks the Save button.

For FieldEdit error messages running in deferred mode, PeopleTools changes the field to red and positions the cursor to the field in error when it displays the message. This allows the user to associate the error message with a specific field.

For warning messages, however, PeopleTools does not change the field to red or position the cursor. For a user to clearly understand to which field a warning message applies, ensure that warning messages clearly describe the fields affected by the warning.

For example, the warning message “Date out of range” would be confusing if there are seven date fields on the page, since a user could not easily determine which date field needed to be reviewed. Instead, you could include bind variables in the message to show which dates are out of range.

Using the Fastest Algorithm

You should determine which algorithms perform the best and have the smallest elapsed time. Tracing does not provide subsecond level of timing information. Plus, tracing imposes a higher overhead to the runtime environment, which skews the elapsed time reading.

However, you can use the %PerfTime system variable for determining elapsed time. %PerfTime retrieves the local system clock time by making a system call, and the return time is down to the millisecond.

The following example of %PerfTime determines how long a program takes to execute:

```
 &Start = %PerfTime;
  &results = "";
  For &I = 1 To &Count;
     &GnnwgNumber = GetNextNumberWithGapsCommit(QEORDER_DTL.QE_QTY, 999999, 1,⇒
"where QE_ORDER_NBR='GNNWG'");
     &results = &results | " : " | &GnnwgNumber;
  End-For;

  &End = %PerfTime;
  &out = "Count = " | &Count | ", total GNNWG time (s) = " | NumberToString⇒
("%6.3", Value(&End - &Start));
```

Using Better Coding Techniques for Performance

This section discusses how to:

- Run a SQL trace.
- Optimize SQL.
- Use the GetNextNumberWithGaps function.
- Consolidate PeopleCode programs.
- Move PeopleCode to a component or page definition.
- Send messages in the SavePostChange event.
- Use metadata and the RowsetCache class.
- Setting MaxCacheMemory

Running a SQL Trace

Run a SQLTrace and review the transaction for SQL statements that have a long processing time.

The duration column (Dur=) in a SQL trace displays this information. If the duration is greater than 100 milliseconds, there may be an opportunity to make this SQL statement run faster. Work with your database administrator to tune the SQL.

Optimizing SQL

It is better to perform a simple join than to issue two related SQL statements separately.

However, if your transaction requires a very complex SQL statement (for instance, anything that uses correlated subqueries) consider breaking it up into multiple SQL statements. You may get more predictable performance this way.

Using the GetNextNumberWithGaps Function

Many applications use a sequence number as a unique key. The last number used is stored in a common table, and a SQL statement is issued to retrieve the last number used and update the table. This locks the common table until the whole transaction is saved and the unit of work committed.

Instead, consider using the GetNextNumberWithGaps PeopleCode function whenever have gaps in the sequence numbering are acceptable. The function retrieves the last number used, increments it by one, and updates the common table. This is done in a separate unit of work, to minimize the time a database lock is held on the common table.

GetNextNumberWithGaps issues a commit only when issued from the SavePreChange or Workflow event.

Consolidating PeopleCode Programs

Consolidate RowInit PeopleCode into one field within the record. This reduces the number of PeopleCode events that need to be triggered. Fewer PeopleCode programs results in fewer PeopleCode objects to manage. Do the same for RowInsert, SaveEdit, SavePreChange, SavePostChange, and Workflow PeopleCode programs.

Moving PeopleCode to a Component or Page Definition

Analyze transactions and move PeopleCode that is specific to a component from the record definition to the component or page definition. This eliminates the need to execute conditional statements, such as `If %Component =`.

This only helps if you are able to move all the PeopleCode in a program from the record to a component or page, and multiple components access that record.

Sending Messages in the SavePostChange Event

Messages sent online should always be coded in the SavePostChange event. To minimize the time that PeopleTools maintains locks on single-threaded messaging tables, behind-the-scenes logic in the SavePostChange event defers sending the message until just prior to the commit for the transaction.

Using Metadata and the RowsetCache Class

If your application uses data that is common, utilized by a number of users, and yet is fairly static, you may see a performance improvement by using the RowsetCache class.

PeopleTools stores application data in a database cache to increase system performance. The RowsetCache class enables you to access this memory structure, created at runtime, and shared by all users.

Note. Non-base language users may see different performance due to language table considerations.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, “RowsetCache Class”.

Setting MaxCacheMemory

PeopleTools stores application data in a memory cache to increase system performance. However, too large a cache can leave insufficient available memory on your system, which leads to reduced performance.

Use this setting to specify the maximum size of the memory cache. PeopleTools prunes the cache to keep it within the specified size, and places the pruned data in a disk cache instead. Because using a disk cache can also reduce performance, the default setting might not be optimal for your application. You can adjust this setting to achieve the best trade-off between speed and available memory.

See *Enterprise PeopleTools 8.45 PeopleBook: System and Server Administration*, "Setting Application Server Domain Parameters," Cache Settings.

Writing More Efficient Code

Follow these steps to write more efficient PeopleCode:

1. Declare all variables.

   One of the conveniences of PeopleCode is that you do not have to declare your variables before you use them. The variable is assigned a type of ANY, taking on the type of the value it is assigned. However, if you use this feature, you lose type-checking at compile time, which can lead to problems at runtime.

   When you validate or save PeopleCode, watch for autodeclared messages and consider adding declarations to your program.

2. Declare variable types specifically.

   Most of the time, you know a variable's type, so you should declare the variable of that type to begin with.

   For example, if you know that a particular variable is going to be an Integer value, declare it to be Integer in the first place. You can get much better runtime performance. It is particularly effective for loop control variables but since an integer has limited range (up to 9 or 10 digits), it must be used judiciously

3. Watch references.

   In PeopleCode function calls, parameters are passed by reference; a reference to the value is passed instead of the value itself. If you are passing a reference to a complex data structure, such as a rowset object or an array, passing by reference saves significant processing.

   Watch out for unexpected results, though. In the following code, the function Test changes the value of *&Str* after the function call.

```
Function Test(&Par as String)

&Par = "Surprise";
end-function;

Local String &Str = "Hello";
Test(&Str);
/* now &Str has the value "surprise" */
```

4. Put Break statements in your Evaluate statements.

   In an Evaluate statement, the When clauses continue to be evaluated until an End-evaluate or a Break statement is encountered.

If you have an Evaluate statement with a number of When clauses, and you only expect one of them to match, put a Break statement following the likely clause. Otherwise, all the subsequent When clauses are evaluated. Your program is still correct, but it is inefficient at runtime, particularly if you have a large number of When clauses, and the Evaluate statement is in a loop.

5. Govern your state.

   One of the key features in PeopleSoft Pure Internet Architecture is that the application server is stateless. When required, the state of your session is bundled up and exchanged between the application server and the web server.

   For example, on a user interaction, the whole state, including your PeopleCode state, has to be serialized to the web server. Then, once the interaction has completed, that state is deserialized in the application server so that your application can continue.

   To improve efficiency:

   - Watch the size of PeopleCode objects that you create (strings, arrays, and so on) to make sure they are only as big as you need them to be.
   - For user interactions, you might be able to change the logic of your program to minimize the state.

     For example if you are building up a large string (a couple of megabytes) and then performing a user interaction, you might be able to change your program logic to build the string after the interaction.
   - For secondary pages that are infrequently accessed but retrieve lots of data, consider setting *No Auto Select* in the PeopleSoft Application Designer for the grids and scroll areas on the secondary page, to prevent loading the data the secondary page when the page buffers are initially built.

     Then add the necessary Select method to the Activate event for the secondary page to load the data into the grid or scroll area.

6. Isolate common expressions.

   The PeopleCode compiler is not an optimizing compiler, unlike some current compilers for languages such as C++. For example, the PeopleCode compiler does not do common subexpression analysis. So, sometimes, if you have a complicated bit of PeopleCode that is used often, you can isolate the common expression yourself. This can make your code look cleaner, and make your code faster, especially if it is in a loop.

   In this example, notice how the common subexpression is broken out:

```
/*---- For this customer, setup time on B is influenced by
     *---- the machine flavors of A. */
    &r_machine = &rs(&idB.GetRecord(Record.MACHINE_INFO);
    If (&typeA = "F") And (&typeB == "U") Then
        &r_machine.SETUP_TIME.Value = 50;
    Else
        &r_machine.SETUP_TIME.Value = 10;
    End-If;
```

   The compiler has to evaluate each occurrence of the expression, even though it would only execute it once.

   Here's another example. Notice that once &RS and &StartDate are created, they can be used repeatedly in the loop, saving significant processing time.

```
  &RS = GetRowset();
   &StartDate = GetField(PSU_CRS_SESSN.START_DATE).Value;
   For &I = 1 To &RS.ActiveRowCount
      &RecStuEnroll = &RS.GetRow(&I).PSU_STU_ENROLL;
```

```
      &Course = &RecStuEnroll.COURSE;
      &Status = &RecStuEnroll.ENROLL_STATUS;
      &PreReqStart = &RS.GetRow(&I).PSU_CRS_SESSN.START_DATE.Value;
      If &Course.Value = "1002" And
            (&Status.Value = "ENR" Or
               &Status.Value = "CMP") Then
         If &PreReqStart < &StartDate Then
            &Completed = True;
            Break;
         End-If;
      End-If;
   End-For;
```

7. Avoid implicit conversions.

   The most common implicit conversion is from a character string to a number and vice versa. You might not be able to do anything about this, but by being aware of it, you might be able to spot opportunities for performance improvement.

   In the following example, two character strings are converted into numeric values before the difference is taken. If this code was in a loop, and one of the values did not change, there would be a significant performance gain to do the conversion once, as the second statement illustrates.

```
  &Diff = &R1.QE_EMPLID.Value - &R2.QE_EMPID.Value;
&Original = &R1.QE_EMPLID.Value;
. . .
&Diff = &Original - &R2.QE_EMPID.Value;
```

8. Choose the right SQL style.

   In certain cases, use SQLExec, as it only returns a single row. In other cases, you could benefit greatly by using a SQL object instead, especially if you can plan to execute a statement more than once with different bind parameters. The performance gain comes from compiling the statement once and executing it many times.

   For instance, code that uses SQLExec might look like this:

```
While (some condition)
    . . .set up &Rec
    SQLExec("%Insert(:1)", &rec);
/* this does a separate tools parse of the sql and db compile
of the statement and execute each time */
End-while;
```

   The following code rewrites the previous example to use the new SQL object:

```
Local SQL &SQL = CreateSQL("%Insert(:1)");
While (some condition)
. . .Setup &Rec
&Sql.Execute(&Rec); /* saves the tools parse and db compile
on the SQL statement and the db setup for the statement */
end-while;
```

SQL objects also have the ReuseCursor property, which can be used for further performance gains.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode API Reference*, "SQL Class," ReuseCursor.

9. Tighten up loops.

   Examine loops to see if code can be placed outside the loop.

   For example, if you are working with file objects and your file layout does not change, there is no reason to set the file layout every time you go through the loop reading lines from the file. Set the file layout once, outside the loop.

10. Set objects to NULL when they will no longer be accessed.

    Once you are finished with an object reference, especially one with a global or component scope, assign it to NULL to get rid of the object. This allows the runtime environment to clean up unused objects, reducing the size of your PeopleCode state.

11. Improve your application classes

    Simple properties (without get/set) are much more efficient than method calls. Be clear in your design what need to be simple properties, properties with get/set, and methods. Never make something a method that really should be a property.

    Analyze your use of properties implemented with get/set. While PeopleCode properties are in a sense "first class" properties with more flexibility in that you can run PeopleCode to actually get and set their values, make sure you actually need get and set methods. If all you have is a normal property which is more of an instance variable then avoid get/set methods. In the example below (without the strikethrough!) by having get/set for the property SomeString you have made it much more inefficient to get/set that property since every property reference has to run some PeopleCode. Often this can creep in when properties are designed to be flexible at the beginning and never subsequently analyzed for whether getters/setters were really needed after all.

```
class Test
...
property String SomeString get set;

end-class;

get SomeString
return &SomeString;
end-get;

set SomeString
&SomeString = &NewValue;
end-set;
```

Writing More Efficient Code Examples

The following are examples of writing more efficient code.

- Beware of the rowset Fill method. This can also be called "What not to do in a PeopleSoft Application Engine PeopleCode step."

  Sometimes you need to examine the algorithm you're using. The following example is a PeopleCode program that adopts this approach: read all the data into a rowset, process it row by row, then update as necessary. One of the reasons this is a bad approach is because you lose the general advantage of set-based programming that you get with PeopleSoft Application Engine programs.

```
Local Rowset &RS;
Local Record &REC;
Local SQL &SQL_UPDATE;

&REC_NAME1 = "Record." | SOME_AET.SOME_TMP;
&RS = CreateRowset(@(&REC_NAME1));
&LINE_NO = 1;


&NUM_ROWS = &RS.Fill("WHERE PROCESS_INSTANCE = :1 AND BUSINESS_UNIT = :2 AND⇒
 TRANSACTION_GROUP = :3 AND ADJUST_TYPE = :4 ", SOME_AET.PROCESS_INSTANCE, SOME_⇒
AET.BUSINESS_UNIT, SOME_AET.TRANSACTION_GROUP, SOME_AET.ADJUST_TYPE);


For &I = 1 To &NUM_ROWS
   &REC = &RS(&I).GetRecord(@(&REC_NAME1));
   &REC.SOME_FIELD.Value = &LINE_NO;
   &REC.Update();
   &LINE_NO = &LINE_NO + 2;
End-For;
```

This code has the following problems:

- There is a chance you will run out of memory in the Fill method if the Select gathers a large amount of data.
- The Fill is selecting all the columns in the table when all that is being updated is on column.

You can change this code to read in the data one row at a time using a SQL object or using a similar algorithm but chunking the rowsets into a manageable size through the use of an appropriate Where clause.

The following are some approximate numbers you can use to see how large a rowset can grow. The overhead for a field buffer (independent of any field data) is approximately 88 bytes. The overhead for a record buffer is approximately 44 bytes. The overhead for a row is approximately 26 bytes. So a rowset with just one record (row) the general approximate formula is as follows:

memory_amount = nrows * (row overhead + nrecords * (rec overhead + nfields * (field overhead) + average cummulative fielddata for all fields))

- The following are some code examples to show isolating common expressions.

  In this example, a simple evaluation goes from happening three times to just once– `&RS_Level2(&I).PSU_TASK_EFFORT`. In addtion, the rewritten code is easier to read.

  Example of code before being rewritten:

```
Local Rowset &RS_Level2;

Local Boolean &TrueOrFalse = (PSU_TASK_RSRC.COMPLETED_FLAG.Value = "N");

For &I = 1 To &RS_Level2.ActiveRowCount
   &RS_Level2(&I).PSU_TASK_EFFORT.EFFORT_DT.Enabled = &TrueOrFalse;
   &RS_Level2(&I).PSU_TASK_EFFORT.EFFORT_AMT.Enabled = &TrueOrFalse;
   &RS_Level2(&I).PSU_TASK_EFFORT.CHARGE_BACK.Enabled = &TrueOrFalse;
End-For;
```

  Example of code after being rewritten:

```
Local Boolean &TrueOrFalse = (PSU_TASK_RSRC.COMPLETED_FLAG.Value = "N");

For &I = 1 To &RS_Level2.ActiveRowCount
   Local Record &TaskEffort = &RS_Level2(&I).PSU_TASK_EFFORT;

   &TaskEffort.EFFORT_DT.Enabled = &TrueOrFalse;
   &TaskEffort.EFFORT_AMT.Enabled = &TrueOrFalse;
   &TaskEffort.CHARGE_BACK.Enabled = &TrueOrFalse;
End-For;
```

In the next example, the following improvements are made to the code:

Shorthand is used: &ThisRs(&J) instead of &ThisRs.GetRow(&J).

Eliminated all the auto-declared messages by declaring all the local variables. This can improve your logic, and possibly give you better performance.

Notice the integer declaration. If you know your variables will fit in an integer (or a float) declare them that way. Runtime performance for Integers can be better than for variables declared as Number.

Fewer evaluation expressions.

Example of code before being rewritten:

```
Local Row &CurrentRow;
&TrueOrFalse = (GetField().Value = "N");
&CurrentRow = GetRow();
For &I = 1 To &CurrentRow.ChildCount
   For &J = 1 To &CurrentRow.GetRowset(&I).ActiveRowCount
      For &K = 1 To &CurrentRow.GetRowset(&I).GetRow(&J).RecordCount
         For &L = 1 To &CurrentRow.GetRowset(&I).GetRow(&J).GetRecord(&K).FieldCount
            &CurrentRow.GetRowset(&I).GetRow(&J).GetRecord(&K).GetField(&L).Enabled⇒
 = &TrueOrFalse;
         End-For;
      End-For;
   End-For;
End-For;
```

Example of code after being rewritten:

```
Local Row &CurrentRow;
Local integer &I, &J, &K, &L;

Local boolean &TrueOrFalse = (GetField().Value = "N");
&CurrentRow = GetRow();
For &i = 1 To &CurrentRow.ChildCount
/*   No specific RowSet, Record, or Field is mentioned!    */
   Local Rowset &ThisRs = &CurrentRow.GetRowset(&i);

   For &J = 1 To &ThisRs.ActiveRowCount
      Local Row &ThisRow = &ThisRs(&J);

      For &K = 1 To &ThisRow.RecordCount
         Local Record &ThisRec = &ThisRow.GetRecord(&K);

         For &L = 1 To &ThisRec.FieldCount
```

```
            &ThisRec.GetField(&L).Enabled = &TrueOrFalse;
         End-For;
      End-For;
   End-For;
End-For;
```

- Concatenating a large number of strings into a large string. Sometimes you need to do this.

  The simplest approach is to do something like:

  ```
  &NewString = &NewString | &NewPiece;
  ```

  In itself this is not a bad approach but you can do this much more efficiently using an application class below.

```
class StringBuffer
   method StringBuffer(&InitialValue As string);
   method Append(&New As string) returns StringBuffer; // allows &X.Append⇒
("this").Append("that").Append("and this")
   method Reset();
   property string Value get set;
   property integer Length readonly;
   property integer MaxLength;
private
   instance array of string &Pieces;
end-class;

method StringBuffer
   /+ &InitialValue as String, +/

   &Pieces = CreateArray(&InitialValue);
   &MaxLength = 2147483647; // default maximum size
   &Length = Len(&InitialValue);
end-method;

method Reset
   &Pieces.Len = 0;
   &Length = 0;
end-method;

method Append
   /+ &New as String +/
   Local integer &TempLength = &Length + Len(&New);
   If &Length > &MaxLength Then
      throw CreateException(0, 0, "Maximum size of StringBuffer exceeded(" | &Max⇒
Length | ")");
   End-If;
   &Length = &TempLength;
   &Pieces.Push(&New);
   return %This;
end-method;

get Value
   /+ Returns String +/
```

```
   Local string &Temp = &Pieces.Join("", "", "", &Length);
   /* collapse array now */
   &Pieces.Len = 1;
   &Pieces[1] = &Temp; /* start out with this combo string */
   Return &Temp;
end-get;

set Value
   /+ &NewValue as String +/
   /* Ditch our current value */
   &Pieces.Len = 1;
   &Pieces[1] = &NewValue; /* start out with this string */
   &Length = Len(&NewValue);
end-set;
```

Use this code as follows:

```
Local StringBuffer &S = create StringBuffer("");

....
&S.Append(&line);

/* to get the value of string simply use &S.Value */
```

Searching PeopleCode for SQL Injection

SQL injection is a technique that enables users to pass unintended SQL to an application. SQL injection is usually caused by developers who use string-building techniques to generate SQL that is subsequently executed.

PeopleSoft recommends you search your PeopleCode for SQL injection vulnerabilities.

To search for potential SQL injection vulnerabilities:

1. Open PeopleSoft Application Designer.
2. Select Edit, Find In. .
3. From the Find In dialog box, select *SQL Injection in PeopleCode* as the find type.

   This only finds potential vulnerabilities.
4. Review flagged PeopleCode programs.

   Vulnerable PeopleCode programs allow unvalidated user input concatenated to SQL.

   See Chapter 16, “Debugging Your Application,” Using the Find In Feature, page 269.

The following functions and methods provide a way for SQL to be submitted to the database, and are therefore subject to SQL injection vulnerabilities:

- SQLExec function
- CreateSQL function
- Rowset class Select method
- Rowset class SelectNew method

- Rowset class Fill method
- Rowset class FillAppend method

Look at the following PeopleCode as an example:

```
rem Retrieve user input from the name field;
&UserInput = GetField(Field.NAME).Value;
SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='"
| &UserInput | "'", &Name, &Phone);
```

The code is meant to enable the user to type in a name and get the person's phone number. In the example, the developer expects that the user will input data such as *Smith*, in which case the resulting SQL would look like this:

```
SELECT NAME, PHONE FROM PS_INFO WHERE NAME='Smith'
```

However, if the user specified *"Smith' OR AGE > 55 --"*, the resulting SQL would look like this:

```
SELECT NAME, PHONE FROM PS_INFO WHERE NAME='Smith' OR AGE > 55 --'
```

Note the use of the comment operator (--) to ignore the trailing single quotation mark placed by the developer's code. This would allow a devious user to find everyone older than 55.

Preventing SQL Injection

Use the following approaches to avoid SQL injection vulnerabilities:

- Where possible, avoid using string-building techniques to generate SQL.

  Note. This cannot always be avoided. String-building does not pose a threat unless unvalidated user input is concatenated to SQL.

- Use bind variables where possible rather that string concatenation.

  The following example is vulnerable:

  ```
  SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='" |
  &UserInput | "'", &Name, &Phone);
  ```

- Use the Quote PeopleCode function on the user input before concatenating it to SQL.

  This pairs the quotation marks in the user input, effectively negating any SQL injection attack.

  The following example is vulnerable:

  ```
  SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='" |
  &UserInput | "'", &Name, &Phone);
  ```

  This example is not vulnerable:

  ```
  SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='" |
  Quote(&UserInput) | "'", &Name, &Phone);
  ```

- Specify whether SQL errors are displayed to the user with the Suppress SQL Error setting in the PSTOOLS section of the application server configuration file. Normally the SQL in error is displayed to the user in a number of messages. If you consider this a security issue, add the following line to your application server config file:

  ```
  Suppress SQL Error=1
  ```

When this is set, any SQL errors do not display details, but refer the user to consult the system log. The detail that was in the SQL message are written to the log file.

See Also

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

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

APPENDIX A

ISO Country and Currency Codes

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

This appendix discusses:

- ISO country codes.
- ISO currency codes.

See Also

"About This PeopleBook." Typographical Conventions and Visual Cues

ISO Country Codes

This table lists the ISO country codes that may appear as country identifiers in PeopleBooks:

ISO Country Code	Country Name
ABW	Aruba
AFG	Afghanistan
AGO	Angola
AIA	Anguilla
ALB	Albania
AND	Andorra
ANT	Netherlands Antilles
ARE	United Arab Emirates
ARG	Argentina
ARM	Armenia
ASM	American Samoa
ATA	Antarctica

ISO Country Code	Country Name
ATF	French Southern Territories
ATG	Antigua and Barbuda
AUS	Australia
AUT	Austria
AZE	Azerbaijan
BDI	Burundi
BEL	Belgium
BEN	Benin
BFA	Burkina Faso
BGD	Bangladesh
BGR	Bulgaria
BHR	Bahrain
BHS	Bahamas
BIH	Bosnia and Herzegovina
BLR	Belarus
BLZ	Belize
BMU	Bermuda
BOL	Bolivia
BRA	Brazil
BRB	Barbados
BRN	Brunei Darussalam
BTN	Bhutan
BVT	Bouvet Island
BWA	Botswana
CAF	Central African Republic
CAN	Canada
CCK	Cocos (Keeling) Islands

ISO Country Code	Country Name
CHE	Switzerland
CHL	Chile
CHN	China
CIV	Cote D'Ivoire
CMR	Cameroon
COD	Congo, The Democratic Republic
COG	Congo
COK	Cook Islands
COL	Colombia
COM	Comoros
CPV	Cape Verde
CRI	Costa Rica
CUB	Cuba
CXR	Christmas Island
CYM	Cayman Islands
CYP	Cyprus
CZE	Czech Republic
DEU	Germany
DJI	Djibouti
DMA	Dominica
DNK	Denmark
DOM	Dominican Republic
DZA	Algeria
ECU	Ecuador
EGY	Egypt
ERI	Eritrea
ESH	Western Sahara

ISO Country Code	Country Name
ESP	Spain
EST	Estonia
ETH	Ethiopia
FIN	Finland
FJI	Fiji
FLK	Falkland Islands (Malvinas)
FRA	France
FRO	Faroe Islands
FSM	Micronesia, Federated States
GAB	Gabon
GBR	United Kingdom
GEO	Georgia
GHA	Ghana
GIB	Gibraltar
GIN	Guinea
GLP	Guadeloupe
GMB	Gambia
GNB	Guinea-Bissau
GNQ	Equatorial Guinea
GRC	Greece
GRD	Grenada
GRL	Greenland
GTM	Guatemala
GUF	French Guiana
GUM	Guam
GUY	Guyana
GXA	GXA - GP Core Country

ISO Country Code	Country Name
GXB	GXB - GP Core Country
GXC	GXC - GP Core Country
GXD	GXD - GP Core Country
HKG	Hong Kong
HMD	Heard and McDonald Islands
HND	Honduras
HRV	Croatia
HTI	Haiti
HUN	Hungary
IDN	Indonesia
IND	India
IOT	British Indian Ocean Territory
IRL	Ireland
IRN	Iran (Islamic Republic Of)
IRQ	Iraq
ISL	Iceland
ISR	Israel
ITA	Italy
JAM	Jamaica
JOR	Jordan
JPN	Japan
KAZ	Kazakstan
KEN	Kenya
KGZ	Kyrgyzstan
KHM	Cambodia
KIR	Kiribati
KNA	Saint Kitts and Nevis

ISO Country Code	Country Name
KOR	Korea, Republic of
KWT	Kuwait
LAO	Lao People’s Democratic Rep
LBN	Lebanon
LBR	Liberia
LBY	Libyan Arab Jamahiriya
LCA	Saint Lucia
LIE	Liechtenstein
LKA	Sri Lanka
LSO	Lesotho
LTU	Lithuania
LUX	Luxembourg
LVA	Latvia
MAC	Macao
MAR	Morocco
MCO	Monaco
MDA	Moldova, Republic of
MDG	Madagascar
MDV	Maldives
MEX	Mexico
MHL	Marshall Islands
MKD	Fmr Yugoslav Rep of Macedonia
MLI	Mali
MLT	Malta
MMR	Myanmar
MNG	Mongolia
MNP	Northern Mariana Islands

ISO Country Code	Country Name
MOZ	Mozambique
MRT	Mauritania
MSR	Montserrat
MTQ	Martinique
MUS	Mauritius
MWI	Malawi
MYS	Malaysia
MYT	Mayotte
NAM	Namibia
NCL	New Caledonia
NER	Niger
NFK	Norfolk Island
NGA	Nigeria
NIC	Nicaragua
NIU	Niue
NLD	Netherlands
NOR	Norway
NPL	Nepal
NRU	Nauru
NZL	New Zealand
OMN	Oman
PAK	Pakistan
PAN	Panama
PCN	Pitcairn
PER	Peru
PHL	Philippines
PLW	Palau

ISO Country Code	Country Name
PNG	Papua New Guinea
POL	Poland
PRI	Puerto Rico
PRK	Korea, Democratic People's Rep
PRT	Portugal
PRY	Paraguay
PSE	Palestinian Territory, Occupie
PYF	French Polynesia
QAT	Qatar
REU	Reunion
ROU	Romania
RUS	Russian Federation
RWA	Rwanda
SAU	Saudi Arabia
SDN	Sudan
SEN	Senegal
SGP	Singapore
SGS	Sth Georgia & Sth Sandwich Is
SHN	Saint Helena
SJM	Svalbard and Jan Mayen
SLB	Solomon Islands
SLE	Sierra Leone
SLV	El Salvador
SMR	San Marino
SOM	Somalia
SPM	Saint Pierre and Miquelon
STP	Sao Tome and Principe

ISO Country Code	Country Name
SUR	Suriname
SVK	Slovakia
SVN	Slovenia
SWE	Sweden
SWZ	Swaziland
SYC	Seychelles
SYR	Syrian Arab Republic
TCA	Turks and Caicos Islands
TCD	Chad
TGO	Togo
THA	Thailand
TJK	Tajikistan
TKL	Tokelau
TKM	Turkmenistan
TLS	East Timor
TON	Tonga
TTO	Trinidad and Tobago
TUN	Tunisia
TUR	Turkey
TUV	Tuvalu
TWN	Taiwan, Province of China
TZA	Tanzania, United Republic of
UGA	Uganda
UKR	Ukraine
UMI	US Minor Outlying Islands
URY	Uruguay
USA	United States

ISO Country Code	Country Name
UZB	Uzbekistan
VAT	Holy See (Vatican City State)
VCT	St Vincent and the Grenadines
VEN	Venezuela
VGB	Virgin Islands (British)
VIR	Virgin Islands (U.S.)
VNM	Viet Nam
VUT	Vanuatu
WLF	Wallis and Futuna Islands
WSM	Samoa
YEM	Yemen
YUG	Yugoslavia
ZAF	South Africa
ZMB	Zambia
ZWE	Zimbabwe

ISO Currency Codes

This table lists the ISO country codes that may appear as currency identifiers in PeopleBooks:

ISO Currency Code	Description
ADP	Andorran Peseta
AED	United Arab Emirates Dirham
AFA	Afghani
AFN	Afghani
ALK	Old Lek
ALL	Lek
AMD	Armenian Dram

ISO Currency Code	Description
ANG	Netherlands Antilles Guilder
AOA	Kwanza
AOK	Kwanza
AON	New Kwanza
AOR	Kwanza Reajustado
ARA	Austral
ARP	Peso Argentino
ARS	Argentine Peso
ARY	Peso
ATS	Schilling
AUD	Australian Dollar
AWG	Aruban Guilder
AZM	Azerbaijanian Manat
BAD	Dinar
BAM	Convertible Marks
BBD	Barbados Dollar
BDT	Taka
BEC	Convertible Franc
BEF	Belgian Franc
BEL	Financial Belgian Franc
BGJ	Lev A/52
BGK	Lev A/62
BGL	Lev
BGN	Bulgarian LEV
BHD	Bahraini Dinar
BIF	Burundi Franc
BMD	Bermudian Dollar

ISO Currency Code	Description
BND	Brunei Dollar
BOB	Boliviano
BOP	Peso
BOV	Mvdol
BRB	Cruzeiro
BRC	Cruzado
BRE	Cruzeiro
BRL	Brazilian Real
BRN	New Cruzado
BRR	Brazilian Real Dollar
BSD	Bahamian Dollar
BTN	Ngultrum
BUK	N/A
BWP	Pula
BYB	Belarussian Ruble
BYR	Belarussian Ruble
BZD	Belize Dollar
CAD	Canadian Dollar
CDF	Franc Congolais
CHF	Swiss Franc
CLF	Unidades de fomento
CLP	Chilean Peso
CNX	Peoples Bank Dollar
CNY	Yuan Renminbi
COP	Colombian Peso
CRC	Costa Rican Colon
CSD	Serbia Dinar

ISO Currency Code	Description
CSJ	Krona A/53
CSK	Koruna
CUP	Cuban Peso
CVE	Cape Verde Escudo
CYP	Cyprus Pound
CZK	Czech Koruna
DEM	Deutsche Mark
DJF	Djibouti Franc
DKK	Danish Krone
DOP	Dominican Peso
DZD	Algerian Dinar
ECS	Sucre
ECV	Unidad de Valor
EEK	Kroon
EGP	Egyptian Pound
EQE	Ekwele
ERN	Nakfa
ESA	Spanish Peseta
ESB	Convertible Peseta
ESP	Spanish Peseta
ETB	Ethiopian Birr
EUR	euro
FIM	Markka
FJD	Fiji Dollar
FKP	Falklands Isl. Pound
FRF	French Franc
GBP	Pound Sterling

ISO Currency Code	Description
GEK	Georgian Coupon
GEL	Lari
GHC	Cedi
GIP	Gibraltar Pound
GMD	Dalasi
GNE	Syli
GNF	Guinea Franc
GNS	Syli
GQE	Ekwele
GRD	Drachma
GTQ	Quetzal
GWE	Guinea Escudo
GWP	Guinea-Bissau Peso
GYD	Guyana Dollar
HKD	Hong Kong Dollar
HNL	Lempira
HRD	Dinar
HRK	Kuna
HTG	Gourde
HUF	Forint
IDR	Rupiah
IEP	Irish Pound
ILP	Pound
ILR	Old Shekel
ILS	New Israeli Sheqel
INR	Indian Rupee
IQD	Iraqi Dinar

ISO Currency Code	Description
IRR	Iranian Rial
ISJ	Old Krona
ISK	Iceland Krona
ITL	Italian Lira
JMD	Jamaican Dollar
JOD	Jordanian Dinar
JPY	Yen
KES	Kenyan Shilling
KGS	Som
KHR	Riel
KMF	Comoro Franc
KPW	North Korean Won
KRW	Won
KWD	Kuwaiti Dinar
KYD	Cayman Islands dollar
KZT	Tenge
LAJ	Kip Pot Pol
LAK	Kip
LBP	Lebanese Pound
LKR	Sri Lanka Rupee
LRD	Liberian Dollar
LSL	Loti
LSM	Maloti
LTL	Lithuanian Litas
LTT	Talonas
LUC	Convertib Francl
LUF	Luxembourg Franc

ISO Currency Code	Description
LUL	Financial Franc
LVL	Latvian Lats
LVR	Latvian Ruble
LYD	Libyan Dinar
MAD	Moroccan Dirham
MAF	Mali Franc
MDL	Moldovan Leu
MGF	Malagasy Franc
MKD	Denar
MLF	Mali Franc
MMK	Kyat
MNT	Tugrik
MOP	Pataca
MRO	Ouguiya
MTL	Maltese Lira
MTP	Maltese Pound
MUR	Mauritius Rupee
MVQ	Maldive Rupee
MVR	Rufiyaa
MWK	Malawian Kwacha
MXN	Mexican Peso
MXP	Mexican Peso
MXV	Mexican UDI
MYR	Malaysian Ringgit
MZE	Mozambique Escudo
MZM	Metical
NAD	Namibia Dollar

ISO Currency Code	Description
NGN	Naira
NIC	Cordoba
NIO	Cordoba Oro
NLG	Netherlands Guilder
NOK	Norwegian Krone
NPR	Nepalese Rupee
NZD	New Zealand Dollar
OMR	Rial Omani
PAB	Balboa
PEI	Inti
PEN	Nuevo Sol
PES	Sol
PGK	Kina
PHP	Philippine Peso
PKR	Pakistan Rupee
PLN	Zloty
PLZ	Zloty
PTE	Portuguese Escudo
PYG	Guarani
QAR	Qatari Rial
ROK	Leu A/52
ROL	Leu
RUB	Russian Ruble
RUR	Russian Federation Rouble
RWF	Rwanda Franc
SAR	Saudi Riyal
SBD	Solomon Islands

ISO Currency Code	Description
SCR	Seychelles Rupee
SDD	Sudanese Dinar
SDP	Sudanese Pound
SEK	Swedish Krona
SGD	Singapore Dollar
SHP	St Helena Pound
SIT	Tolar
SKK	Slovak Koruna
SLL	Leone
SOS	Somali Shilling
SRG	Surinam Guilder
STD	Dobra
SUR	Rouble
SVC	El Salvador Colon
SYP	Syrian Pound
SZL	Lilangeni
THB	Baht
TJR	Tajik Ruble
TJS	Somoni
TMM	Manat
TND	Tunisian Dinar
TOP	Pa’anga
TPE	Timor Escudo
TRL	Turkish Lira
TTD	Trinidad Dollar
TWD	New Taiwan Dollar
TZS	Tanzanian Shilling

ISO Currency Code	Description
UAH	Hryvnia
UAK	Karbovanet
UGS	Uganda Shilling
UGW	Old Shilling
UGX	Uganda Shilling
USD	US Dollar
USN	US Dollar (Next day)
USS	US Dollar (Same day)
UYN	Old Uruguay Peso
UYP	Uruguayan Peso
UYU	Peso Uruguayo
UZS	Uzbekistan Sum
VEB	Bolivar
VNC	Old Dong
VND	Dong
VUV	Vatu
WST	Tala
XAF	CFA Franc BEAC
XAG	Silver
XAU	GOLD
XBA	European Composite Unit
XBB	European Monetary Unit
XBC	European Unit of Account 9
XBD	European Unit of Account 17
XCD	East Caribbean Dollar
XDR	SDR
XEU	EU Currency (E.C.U)

ISO Currency Code	Description
XFO	Gold-Franc
XFU	UIC-Franc
XOF	CFA Franc BCEAO
XPD	Palladium
XPF	CFP Franc
XPT	Platinum
XTS	For Testing Purposes
XXX	Non Currency Transaction
YDD	Yemeni Din
YER	Yemeni Rial
YUD	New Yugoslavian Dinar
YUM	New Dinar
YUN	Yugoslavian Dinar
ZAL	Financial Rand
ZAR	Rand
ZMK	Zambian Kwacha
ZRN	New Zaire
ZRZ	Zaire
ZWC	Rhodesian Dollar
ZWD	Zimbabwe Dollar

Glossary of PeopleSoft Terms

absence entitlement This element defines rules for granting paid time off for valid absences, such as sick time, vacation, and maternity leave. An absence entitlement element defines the entitlement amount, frequency, and entitlement period.

absence take This element defines the conditions that must be met before a payee is entitled to take paid time off.

accounting class In PeopleSoft Enterprise Performance Management, the accounting class defines how a resource is treated for generally accepted accounting practices. The Inventory class indicates whether a resource becomes part of a balance sheet account, such as inventory or fixed assets, while the Non-inventory class indicates that the resource is treated as an expense of the period during which it occurs.

accounting date The accounting date indicates when a transaction is recognized, as opposed to the date the transaction actually occurred. The accounting date and transaction date can be the same. The accounting date determines the period in the general ledger to which the transaction is to be posted. You can only select an accounting date that falls within an open period in the ledger to which you are posting. The accounting date for an item is normally the invoice date.

accounting split The accounting split method indicates how expenses are allocated or divided among one or more sets of accounting ChartFields.

accumulator You use an accumulator to store cumulative values of defined items as they are processed. You can accumulate a single value over time or multiple values over time. For example, an accumulator could consist of all voluntary deductions, or all company deductions, enabling you to accumulate amounts. It allows total flexibility for time periods and values accumulated.

action reason The reason an employee's job or employment information is updated. The action reason is entered in two parts: a personnel action, such as a promotion, termination, or change from one pay group to another—and a reason for that action. Action reasons are used by PeopleSoft Human Resources, PeopleSoft Benefits Administration, PeopleSoft Stock Administration, and the COBRA Administration feature of the Base Benefits business process.

action template In PeopleSoft Receivables, outlines a set of escalating actions that the system or user performs based on the period of time that a customer or item has been in an action plan for a specific condition.

activity In PeopleSoft Enterprise Learning Management, an instance of a catalog item (sometimes called a class) that is available for enrollment. The activity defines such things as the costs that are associated with the offering, enrollment limits and deadlines, and waitlisting capacities.

In PeopleSoft Enterprise Performance Management, the work of an organization and the aggregation of actions that are used for activity-based costing.

In PeopleSoft Project Costing, the unit of work that provides a further breakdown of projects—usually into specific tasks.

In PeopleSoft Workflow, a specific transaction that you might need to perform in a business process. Because it consists of the steps that are used to perform a transaction, it is also known as a step map.

agreement In PeopleSoft eSettlements, provides a way to group and specify processing options, such as payment terms, pay from a bank, and notifications by a buyer and supplier location combination.

allocation rule In PeopleSoft Enterprise Incentive Management, an expression within compensation plans that enables the system to assign transactions to nodes and participants. During transaction allocation, the allocation engine traverses the compensation structure from the current node to the root node, checking each node for plans that contain allocation rules.

alternate account A feature in PeopleSoft General Ledger that enables you to create a statutory chart of accounts and enter statutory account transactions at the detail transaction level, as required for recording and reporting by some national governments.

AR specialist Abbreviation for *receivables specialist.* In PeopleSoft Receivables, an individual in who tracks and resolves deductions and disputed items.

arbitration plan In PeopleSoft Enterprise Pricer, defines how price rules are to be applied to the base price when the transaction is priced.

assessment rule In PeopleSoft Receivables, a user-defined rule that the system uses to evaluate the condition of a customer's account or of individual items to determine whether to generate a follow-up action.

asset class An asset group used for reporting purposes. It can be used in conjunction with the asset category to refine asset classification.

attribute/value pair In PeopleSoft Directory Interface, relates the data that makes up an entry in the directory information tree.

authentication server A server that is set up to verify users of the system.

base time period In PeopleSoft Business Planning, the lowest level time period in a calendar.

benchmark job In PeopleSoft Workforce Analytics, a benchmark job is a job code for which there is corresponding salary survey data from published, third-party sources.

book In PeopleSoft Asset Management, used for storing financial and tax information, such as costs, depreciation attributes, and retirement information on assets.

branch A tree node that rolls up to nodes above it in the hierarchy, as defined in PeopleSoft Tree Manager.

budgetary account only An account used by the system only and not by users; this type of account does not accept transactions. You can only budget with this account. Formerly called "system-maintained account."

budget check In commitment control, the processing of source transactions against control budget ledgers, to see if they pass, fail, or pass with a warning.

budget control In commitment control, budget control ensures that commitments and expenditures don't exceed budgets. It enables you to track transactions against corresponding budgets and terminate a document's cycle if the defined budget conditions are not met. For example, you can prevent a purchase order from being dispatched to a vendor if there are insufficient funds in the related budget to support it.

budget period The interval of time (such as 12 months or 4 quarters) into which a period is divided for budgetary and reporting purposes. The ChartField allows maximum flexibility to define operational accounting time periods without restriction to only one calendar.

business event In PeopleSoft Receivables, defines the processing characteristics for the Receivable Update process for a draft activity.

	In PeopleSoft Sales Incentive Management, an original business transaction or activity that may justify the creation of a PeopleSoft Enterprise Incentive Management event (a sale, for example).
business unit	A corporation or a subset of a corporation that is independent with regard to one or more operational or accounting functions.
buyer	In PeopleSoft eSettlements, an organization (or business unit, as opposed to an individual) that transacts with suppliers (vendors) within the system. A buyer creates payments for purchases that are made in the system.
catalog item	In PeopleSoft Enterprise Learning Management, a specific topic that a learner can study and have tracked. For example, “Introduction to Microsoft Word.” A catalog item contains general information about the topic and includes a course code, description, categorization, keywords, and delivery methods. A catalog item can have one or more learning activities.
catalog map	In PeopleSoft Catalog Management, translates values from the catalog source data to the format of the company’s catalog.
catalog partner	In PeopleSoft Catalog Management, shares responsibility with the enterprise catalog manager for maintaining catalog content.
categorization	Associates partner offerings with catalog offerings and groups them into enterprise catalog categories.
channel	In PeopleSoft MultiChannel Framework, email, chat, voice (computer telephone integration [CTI]), or a generic event.
ChartField	A field that stores a chart of accounts, resources, and so on, depending on the PeopleSoft application. ChartField values represent individual account numbers, department codes, and so forth.
ChartField balancing	You can require specific ChartFields to match up (balance) on the debit and the credit side of a transaction.
ChartField combination edit	The process of editing journal lines for valid ChartField combinations based on user-defined rules.
ChartKey	One or more fields that uniquely identify each row in a table. Some tables contain only one field as the key, while others require a combination.
checkbook	In PeopleSoft Promotions Management, enables you to view financial data (such as planned, incurred, and actual amounts) that is related to funds and trade promotions.
Class ChartField	A ChartField value that identifies a unique appropriation budget key when you combine it with a fund, department ID, and program code, as well as a budget period. Formerly called *sub-classification.*
clone	In PeopleCode, to make a unique copy. In contrast, to *copy* may mean making a new reference to an object, so if the underlying object is changed, both the copy and the original change.
collection	To make a set of documents available for searching in Verity, you must first create at least one collection. A collection is set of directories and files that allow search application users to use the Verity search engine to quickly find and display source documents that match search criteria. A collection is a set of statistics and pointers to the source documents, stored in a proprietary format on a file server. Because a collection can only store information for a single location, PeopleSoft maintains a set of collections (one per language code) for each search index object.

collection rule In PeopleSoft Receivables, a user-defined rule that defines actions to take for a customer based on both the amount and the number of days past due for outstanding balances.

compensation object In PeopleSoft Enterprise Incentive Management, a node within a compensation structure. Compensation objects are the building blocks that make up a compensation structure's hierarchical representation.

compensation structure In PeopleSoft Enterprise Incentive Management, a hierarchical relationship of compensation objects that represents the compensation-related relationship between the objects.

condition In PeopleSoft Receivables, occurs when there is a change of status for a customer's account, such as reaching a credit limit or exceeding a user-defined balance due.

configuration parameter catalog Used to configure an external system with PeopleSoft. For example, a configuration parameter catalog might set up configuration and communication parameters for an external server.

configuration plan In PeopleSoft Enterprise Incentive Management, configuration plans hold allocation information for common variables (not incentive rules) and are attached to a node without a participant. Configuration plans are not processed by transactions.

content reference Content references are pointers to content registered in the portal registry. These are typically either URLs or iScripts. Content references fall into three categories: target content, templates, and template pagelets.

context In PeopleCode, determines which buffer fields can be contextually referenced and which is the current row of data on each scroll level when a PeopleCode program is running.

In PeopleSoft Enterprise Incentive Management, a mechanism that is used to determine the scope of a processing run. PeopleSoft Enterprise Incentive Management uses three types of context: plan, period, and run-level.

control table Stores information that controls the processing of an application. This type of processing might be consistent throughout an organization, or it might be used only by portions of the organization for more limited sharing of data.

cost profile A combination of a receipt cost method, a cost flow, and a deplete cost method. A profile is associated with a cost book and determines how items in that book are valued, as well as how the material movement of the item is valued for the book.

cost row A cost transaction and amount for a set of ChartFields.

current learning In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's in-progress learning activities and programs.

data acquisition In PeopleSoft Enterprise Incentive Management, the process during which raw business transactions are acquired from external source systems and fed into the operational data store (ODS).

data elements Data elements, at their simplest level, define a subset of data and the rules by which to group them.

For Workforce Analytics, data elements are rules that tell the system what measures to retrieve about your workforce groups.

dataset A data grouping that enables role-based filtering and distribution of data. You can limit the range and quantity of data that is displayed for a user by associating dataset rules with user roles. The result of dataset rules is a set of data that is appropriate for the user's roles.

delivery method In PeopleSoft Enterprise Learning Management, identifies the primary type of delivery method in which a particular learning activity is offered. Also provides default values for the learning activity, such as cost and language. This is primarily used to help learners search the catalog for the type of delivery from which they learn best. Because PeopleSoft Enterprise Learning Management is a blended learning system, it does not enforce the delivery method.

In PeopleSoft Supply Chain Management, identifies the method by which goods are shipped to their destinations (such as truck, air, rail, and so on). The delivery method is specified when creating shipment schedules.

delivery method type In PeopleSoft Enterprise Learning Management, identifies how learning activities can be delivered—for example, through online learning, classroom instruction, seminars, books, and so forth—in an organization. The type determines whether the delivery method includes scheduled components.

directory information tree In PeopleSoft Directory Interface, the representation of a directory's hierarchical structure.

document sequencing A flexible method that sequentially numbers the financial transactions (for example, bills, purchase orders, invoices, and payments) in the system for statutory reporting and for tracking commercial transaction activity.

dynamic detail tree A tree that takes its detail values—dynamic details—directly from a table in the database, rather than from a range of values that are entered by the user.

edit table A table in the database that has its own record definition, such as the Department table. As fields are entered into a PeopleSoft application, they can be validated against an edit table to ensure data integrity throughout the system.

effective date A method of dating information in PeopleSoft applications. You can predate information to add historical data to your system, or postdate information in order to enter it before it actually goes into effect. By using effective dates, you don't delete values; you enter a new value with a current effective date.

EIM ledger Abbreviation for *Enterprise Incentive Management ledger.* In PeopleSoft Enterprise Incentive Management, an object to handle incremental result gathering within the scope of a participant. The ledger captures a result set with all of the appropriate traces to the data origin and to the processing steps of which it is a result.

elimination set In PeopleSoft General Ledger, a related group of intercompany accounts that is processed during consolidations.

entry event In PeopleSoft General Ledger, Receivables, Payables, Purchasing, and Billing, a business process that generates multiple debits and credits resulting from single transactions to produce standard, supplemental accounting entries.

equitization In PeopleSoft General Ledger, a business process that enables parent companies to calculate the net income of subsidiaries on a monthly basis and adjust that amount to increase the investment amount and equity income amount before performing consolidations.

event A predefined point either in the Component Processor flow or in the program flow. As each point is encountered, the event activates each component, triggering any PeopleCode program that is associated with that component and that event. Examples of events are FieldChange, SavePreChange, and RowDelete.

In PeopleSoft Human Resources, also refers to an incident that affects benefits eligibility.

event propagation process In PeopleSoft Sales Incentive Management, a process that determines, through logic, the propagation of an original PeopleSoft Enterprise Incentive Management event and creates a derivative (duplicate) of the original event to be processed by other objects.

	Sales Incentive Management uses this mechanism to implement splits, roll-ups, and so on. Event propagation determines who receives the credit.
exception	In PeopleSoft Receivables, an item that either is a deduction or is in dispute.
exclusive pricing	In PeopleSoft Order Management, a type of arbitration plan that is associated with a price rule. Exclusive pricing is used to price sales order transactions.
fact	In PeopleSoft applications, facts are numeric data values from fields from a source database as well as an analytic application. A fact can be anything you want to measure your business by, for example, revenue, actual, budget data, or sales numbers. A fact is stored on a fact table.
forecast item	A logical entity with a unique set of descriptive demand and forecast data that is used as the basis to forecast demand. You create forecast items for a wide range of uses, but they ultimately represent things that you buy, sell, or use in your organization and for which you require a predictable usage.
fund	In PeopleSoft Promotions Management, a budget that can be used to fund promotional activity. There are four funding methods: top down, fixed accrual, rolling accrual, and zero-based accrual.
generic process type	In PeopleSoft Process Scheduler, process types are identified by a generic process type. For example, the generic process type SQR includes all SQR process types, such as SQR process and SQR report.
group	In PeopleSoft Billing and Receivables, a posting entity that comprises one or more transactions (items, deposits, payments, transfers, matches, or write-offs). In PeopleSoft Human Resources Management and Supply Chain Management, any set of records that are associated under a single name or variable to run calculations in PeopleSoft business processes. In PeopleSoft Time and Labor, for example, employees are placed in groups for time reporting purposes.
incentive object	In PeopleSoft Enterprise Incentive Management, the incentive-related objects that define and support the PeopleSoft Enterprise Incentive Management calculation process and results, such as plan templates, plans, results data, user interaction objects, and so on.
incentive rule	In PeopleSoft Sales Incentive Management, the commands that act on transactions and turn them into compensation. A rule is one part in the process of turning a transaction into compensation.
incur	In PeopleSoft Promotions Management, to become liable for a promotional payment. In other words, you owe that amount to a customer for promotional activities.
item	In PeopleSoft Inventory, a tangible commodity that is stored in a business unit (shipped from a warehouse). In PeopleSoft Demand Planning, Inventory Policy Planning, and Supply Planning, a noninventory item that is designated as being used for planning purposes only. It can represent a family or group of inventory items. It can have a planning bill of material (BOM) or planning routing, and it can exist as a component on a planning BOM. A planning item cannot be specified on a production or engineering BOM or routing, and it cannot be used as a component in a production. The quantity on hand will never be maintained. In PeopleSoft Receivables, an individual receivable. An item can be an invoice, a credit memo, a debit memo, a write-off, or an adjustment.
KPI	An abbreviation for *key performance indicator.* A high-level measurement of how well an organization is doing in achieving critical success factors. This defines the data value or calculation upon which an assessment is determined.

LDIF file Abbreviation for *Lightweight Directory Access Protocol (LDAP) Data Interchange Format file.* Contains discrepancies between PeopleSoft data and directory data.

learner group In PeopleSoft Enterprise Learning Management, a group of learners who are linked to the same learning environment. Members of the learner group can share the same attributes, such as the same department or job code. Learner groups are used to control access to and enrollment in learning activities and programs. They are also used to perform group enrollments and mass enrollments in the back office.

learning components In PeopleSoft Enterprise Learning Management, the foundational building blocks of learning activities. PeopleSoft Enterprise Learning Management supports six basic types of learning components: web-based, session, webcast, test, survey, and assignment. One or more of these learning component types compose a single learning activity.

learning environment In PeopleSoft Enterprise Learning Management, identifies a set of categories and catalog items that can be made available to learner groups. Also defines the default values that are assigned to the learning activities and programs that are created within a particular learning environment. Learning environments provide a way to partition the catalog so that learners see only those items that are relevant to them.

learning history In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's completed learning activities and programs.

ledger mapping You use ledger mapping to relate expense data from general ledger accounts to resource objects. Multiple ledger line items can be mapped to one or more resource IDs. You can also use ledger mapping to map dollar amounts (referred to as *rates*) to business units. You can map the amounts in two different ways: an actual amount that represents actual costs of the accounting period, or a budgeted amount that can be used to calculate the capacity rates as well as budgeted model results. In PeopleSoft Enterprise Warehouse, you can map general ledger accounts to the EW Ledger table.

library section In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan (or template) and that is available for other plans to share. Changes to a library section are reflected in all plans that use it.

linked section In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan template but appears in a plan. Changes to linked sections propagate to plans using that section.

linked variable In PeopleSoft Enterprise Incentive Management, a variable that is defined and maintained in a plan template and that also appears in a plan. Changes to linked variables propagate to plans using that variable.

load In PeopleSoft Inventory, identifies a group of goods that are shipped together. Load management is a feature of PeopleSoft Inventory that is used to track the weight, the volume, and the destination of a shipment.

local functionality In PeopleSoft HRMS, the set of information that is available for a specific country. You can access this information when you click the appropriate country flag in the global window, or when you access it by a local country menu.

location Locations enable you to indicate the different types of addresses—for a company, for example, one address to receive bills, another for shipping, a third for postal deliveries, and a separate street address. Each address has a different location number. The primary location—indicated by a *1*—is the address you use most often and may be different from the main address.

logistical task In PeopleSoft Services Procurement, an administrative task that is related to hiring a service provider. Logistical tasks are linked to the service type on the work order so that different types of services can have different logistical tasks. Logistical tasks include both preapproval tasks (such as assigning a new badge or ordering a new

	laptop) and postapproval tasks (such as scheduling orientation or setting up the service provider email). The logistical tasks can be mandatory or optional. Mandatory preapproval tasks must be completed before the work order is approved. Mandatory postapproval tasks, on the other hand, must be completed before a work order is released to a service provider.
market template	In PeopleSoft Enterprise Incentive Management, additional functionality that is specific to a given market or industry and is built on top of a product category.
match group	In PeopleSoft Receivables, a group of receivables items and matching offset items. The system creates match groups by using user-defined matching criteria for selected field values.
MCF server	Abbreviation for *PeopleSoft MultiChannel Framework server.* Comprises the universal queue server and the MCF log server. Both processes are started when *MCF Servers* is selected in an application server domain configuration.
merchandising activity	In PeopleSoft Promotions Management, a specific discount type that is associated with a trade promotion (such as off-invoice, billback or rebate, or lump-sum payment) that defines the performance that is required to receive the discount. In the industry, you may know this as an offer, a discount, a merchandising event, an event, or a tactic.
meta-SQL	Meta-SQL constructs expand into platform-specific Structured Query Language (SQL) substrings. They are used in functions that pass SQL strings, such as in SQL objects, the SQLExec function, and PeopleSoft Application Engine programs.
metastring	Metastrings are special expressions included in SQL string literals. The metastrings, prefixed with a percent (%) symbol, are included directly in the string literals. They expand at run time into an appropriate substring for the current database platform.
multibook	In PeopleSoft General Ledger, multiple ledgers having multiple-base currencies that are defined for a business unit, with the option to post a single transaction to all base currencies (all ledgers) or to only one of those base currencies (ledgers).
multicurrency	The ability to process transactions in a currency other than the business unit's base currency.
national allowance	In PeopleSoft Promotions Management, a promotion at the corporate level that is funded by nondiscretionary dollars. In the industry, you may know this as a national promotion, a corporate promotion, or a corporate discount.
node-oriented tree	A tree that is based on a detail structure, but the detail values are not used.
pagelet	Each block of content on the home page is called a pagelet. These pagelets display summary information within a small rectangular area on the page. The pagelet provide users with a snapshot of their most relevant PeopleSoft and non-PeopleSoft content.
participant	In PeopleSoft Enterprise Incentive Management, participants are recipients of the incentive compensation calculation process.
participant object	Each participant object may be related to one or more compensation objects. See also *compensation object.*
partner	A company that supplies products or services that are resold or purchased by the enterprise.
pay cycle	In PeopleSoft Payables, a set of rules that define the criteria by which it should select scheduled payments for payment creation.
pending item	In PeopleSoft Receivables, an individual receivable (such as an invoice, a credit memo, or a write-off) that has been entered in or created by the system, but hasn't been posted.

PeopleCode PeopleCode is a proprietary language, executed by the PeopleSoft application processor. PeopleCode generates results based upon existing data or user actions. By using business interlink objects, external services are available to all PeopleSoft applications wherever PeopleCode can be executed.

PeopleCode event An action that a user takes upon an object, usually a record field, that is referenced within a PeopleSoft page.

PeopleSoft Internet Architecture The fundamental architecture on which PeopleSoft 8 applications are constructed, consisting of a relational database management system (RDBMS), an application server, a web server, and a browser.

performance measurement In PeopleSoft Enterprise Incentive Management, a variable used to store data (similar to an aggregator, but without a predefined formula) within the scope of an incentive plan. Performance measures are associated with a plan calendar, territory, and participant. Performance measurements are used for quota calculation and reporting.

period context In PeopleSoft Enterprise Incentive Management, because a participant typically uses the same compensation plan for multiple periods, the period context associates a plan context with a specific calendar period and fiscal year. The period context references the associated plan context, thus forming a chain. Each plan context has a corresponding set of period contexts.

plan In PeopleSoft Sales Incentive Management, a collection of allocation rules, variables, steps, sections, and incentive rules that instruct the PeopleSoft Enterprise Incentive Management engine in how to process transactions.

plan context In PeopleSoft Enterprise Incentive Management, correlates a participant with the compensation plan and node to which the participant is assigned, enabling the PeopleSoft Enterprise Incentive Management system to find anything that is associated with the node and that is required to perform compensation processing. Each participant, node, and plan combination represents a unique plan context—if three participants are on a compensation structure, each has a different plan context. Configuration plans are identified by plan contexts and are associated with the participants that refer to them.

plan template In PeopleSoft Enterprise Incentive Management, the base from which a plan is created. A plan template contains common sections and variables that are inherited by all plans that are created from the template. A template may contain steps and sections that are not visible in the plan definition.

planned learning In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's planned learning activities and programs.

planning instance In PeopleSoft Supply Planning, a set of data (business units, items, supplies, and demands) constituting the inputs and outputs of a supply plan.

portal registry In PeopleSoft applications, the portal registry is a tree-like structure in which content references are organized, classified, and registered. It is a central repository that defines both the structure and content of a portal through a hierarchical, tree-like structure of folders useful for organizing and securing content references.

price list In PeopleSoft Enterprise Pricer, enables you to select products and conditions for which the price list applies to a transaction. During a transaction, the system either determines the product price based on the predefined search hierarchy for the transaction or uses the product's lowest price on any associated, active price lists. This price is used as the basis for any further discounts and surcharges.

price rule In PeopleSoft Enterprise Pricer, defines the conditions that must be met for adjustments to be applied to the base price. Multiple rules can apply when conditions of each rule are met.

price rule condition In PeopleSoft Enterprise Pricer, selects the price-by fields, the values for the price-by fields, and the operator that determines how the price-by fields are related to the transaction.

price rule key In PeopleSoft Enterprise Pricer, defines the fields that are available to define price rule conditions (which are used to match a transaction) on the price rule.

process category In PeopleSoft Process Scheduler, processes that are grouped for server load balancing and prioritization.

process group In PeopleSoft Financials, a group of application processes (performed in a defined order) that users can initiate in real time, directly from a transaction entry page.

process definition Process definitions define each run request.

process instance A unique number that identifies each process request. This value is automatically incremented and assigned to each requested process when the process is submitted to run.

process job You can link process definitions into a job request and process each request serially or in parallel. You can also initiate subsequent processes based on the return code from each prior request.

process request A single run request, such as a Structured Query Report (SQR), a COBOL or Application Engine program, or a Crystal report that you run through PeopleSoft Process Scheduler.

process run control A PeopleTools variable used to retain PeopleSoft Process Scheduler values needed at runtime for all requests that reference a run control ID. Do not confuse these with application run controls, which may be defined with the same run control ID, but only contain information specific to a given application process request.

product category In PeopleSoft Enterprise Incentive Management, indicates an application in the Enterprise Incentive Management suite of products. Each transaction in the PeopleSoft Enterprise Incentive Management system is associated with a product category.

programs In PeopleSoft Enterprise Learning Management, a high-level grouping that guides the learner along a specific learning path through sections of catalog items. PeopleSoft Enterprise Learning Systems provides two types of programs—curricula and certifications.

progress log In PeopleSoft Services Procurement, tracks deliverable-based projects. This is similar to the time sheet in function and process. The service provider contact uses the progress log to record and submit progress on deliverables. The progress can be logged by the activity that is performed, by the percentage of work that is completed, or by the completion of milestone activities that are defined for the project.

project transaction In PeopleSoft Project Costing, an individual transaction line that represents a cost, time, budget, or other transaction row.

promotion In PeopleSoft Promotions Management, a trade promotion, which is typically funded from trade dollars and used by consumer products manufacturers to increase sales volume.

publishing In PeopleSoft Enterprise Incentive Management, a stage in processing that makes incentive-related results available to participants.

record group A set of logically and functionally related control tables and views. Record groups help enable TableSet sharing, which eliminates redundant data entry. Record groups ensure that TableSet sharing is applied consistently across all related tables and views.

record input VAT flag Abbreviation for *record input value-added tax flag*. Within PeopleSoft Purchasing, Payables, and General Ledger, this flag indicates that you are recording input VAT

	on the transaction. This flag, in conjunction with the record output VAT flag, is used to determine the accounting entries created for a transaction and to determine how a transaction is reported on the VAT return. For all cases within Purchasing and Payables where VAT information is tracked on a transaction, this flag is set to Yes. This flag is not used in PeopleSoft Order Management, Billing, or Receivables, where it is assumed that you are always recording only output VAT, or in PeopleSoft Expenses, where it is assumed that you are always recording only input VAT.
record output VAT flag	Abbreviation for *record output value-added tax flag*. See *record input VAT flag*.
reference data	In PeopleSoft Sales Incentive Management, system objects that represent the sales organization, such as territories, participants, products, customers, channels, and so on.
reference object	In PeopleSoft Enterprise Incentive Management, this dimension-type object further defines the business. Reference objects can have their own hierarchy (for example, product tree, customer tree, industry tree, and geography tree).
reference transaction	In commitment control, a reference transaction is a source transaction that is referenced by a higher-level (and usually later) source transaction, in order to automatically reverse all or part of the referenced transaction's budget-checked amount. This avoids duplicate postings during the sequential entry of the transaction at different commitment levels. For example, the amount of an encumbrance transaction (such as a purchase order) will, when checked and recorded against a budget, cause the system to concurrently reference and relieve all or part of the amount of a corresponding pre-encumbrance transaction, such as a purchase requisition.
regional sourcing	In PeopleSoft Purchasing, provides the infrastructure to maintain, display, and select an appropriate vendor and vendor pricing structure that is based on a regional sourcing model where the multiple ship to locations are grouped. Sourcing may occur at a level higher than the ship to location.
relationship object	In PeopleSoft Enterprise Incentive Management, these objects further define a compensation structure to resolve transactions by establishing associations between compensation objects and business objects.
remote data source data	Data that is extracted from a separate database and migrated into the local database.
REN server	Abbreviation for *real-time event notification server* in PeopleSoft MultiChannel Framework.
requester	In PeopleSoft eSettlements, an individual who requests goods or services and whose ID appears on the various procurement pages that reference purchase orders.
role	Describes how people fit into PeopleSoft Workflow. A role is a class of users who perform the same type of work, such as clerks or managers. Your business rules typically specify what user role needs to do an activity.
role user	A PeopleSoft Workflow user. A person's role user ID serves much the same purpose as a user ID does in other parts of the system. PeopleSoft Workflow uses role user IDs to determine how to route worklist items to users (through an email address, for example) and to track the roles that users play in the workflow. Role users do not need PeopleSoft user IDs.
roll up	In a tree, to roll up is to total sums based on the information hierarchy.
run control	A run control is a type of online page that is used to begin a process, such as the batch processing of a payroll run. Run control pages generally start a program that manipulates data.
run control ID	A unique ID to associate each user with his or her own run control table entries.

run-level context In PeopleSoft Enterprise Incentive Management, associates a particular run (and batch ID) with a period context and plan context. Every plan context that participates in a run has a separate run-level context. Because a run cannot span periods, only one run-level context is associated with each plan context.

search query You use this set of objects to pass a query string and operators to the search engine. The search index returns a set of matching results with keys to the source documents.

section In PeopleSoft Enterprise Incentive Management, a collection of incentive rules that operate on transactions of a specific type. Sections enable plans to be segmented to process logical events in different sections.

security event In commitment control, security events trigger security authorization checking, such as budget entries, transfers, and adjustments; exception overrides and notifications; and inquiries.

serial genealogy In PeopleSoft Manufacturing, the ability to track the composition of a specific, serial-controlled item.

serial in production In PeopleSoft Manufacturing, enables the tracing of serial information for manufactured items. This is maintained in the Item Master record.

session In PeopleSoft Enterprise Learning Management, a single meeting day of an activity (that is, the period of time between start and finish times within a day). The session stores the specific date, location, meeting time, and instructor. Sessions are used for scheduled training.

session template In PeopleSoft Enterprise Learning Management, enables you to set up common activity characteristics that may be reused while scheduling a PeopleSoft Enterprise Learning Management activity—characteristics such as days of the week, start and end times, facility and room assignments, instructors, and equipment. A session pattern template can be attached to an activity that is being scheduled. Attaching a template to an activity causes all of the default template information to populate the activity session pattern.

setup relationship In PeopleSoft Enterprise Incentive Management, a relationship object type that associates a configuration plan with any structure node.

share driver expression In PeopleSoft Business Planning, a named planning method similar to a driver expression, but which you can set up globally for shared use within a single planning application or to be shared between multiple planning applications through PeopleSoft Enterprise Warehouse.

single signon With single signon, users can, after being authenticated by a PeopleSoft application server, access a second PeopleSoft application server without entering a user ID or password.

source transaction In commitment control, any transaction generated in a PeopleSoft or third-party application that is integrated with commitment control and which can be checked against commitment control budgets. For example, a pre-encumbrance, encumbrance, expenditure, recognized revenue, or collected revenue transaction.

SpeedChart A user-defined shorthand key that designates several ChartKeys to be used for voucher entry. Percentages can optionally be related to each ChartKey in a SpeedChart definition.

SpeedType A code representing a combination of ChartField values. SpeedTypes simplify the entry of ChartFields commonly used together.

staging A method of consolidating selected partner offerings with the offerings from the enterprise’s other partners.

statutory account Account required by a regulatory authority for recording and reporting financial results. In PeopleSoft, this is equivalent to the Alternate Account (ALTACCT) ChartField.

step In PeopleSoft Sales Incentive Management, a collection of sections in a plan. Each step corresponds to a step in the job run.

storage level In PeopleSoft Inventory, identifies the level of a material storage location. Material storage locations are made up of a business unit, a storage area, and a storage level. You can set up to four storage levels.

subcustomer qualifier A value that groups customers into a division for which you can generate detailed history, aging, events, and profiles.

Summary ChartField You use summary ChartFields to create summary ledgers that roll up detail amounts based on specific detail values or on selected tree nodes. When detail values are summarized using tree nodes, summary ChartFields must be used in the summary ledger data record to accommodate the maximum length of a node name (20 characters).

summary ledger An accounting feature used primarily in allocations, inquiries, and PS/nVision reporting to store combined account balances from detail ledgers. Summary ledgers increase speed and efficiency of reporting by eliminating the need to summarize detail ledger balances each time a report is requested. Instead, detail balances are summarized in a background process according to user-specified criteria and stored on summary ledgers. The summary ledgers are then accessed directly for reporting.

summary time period In PeopleSoft Business Planning, any time period (other than a base time period) that is an aggregate of other time periods, including other summary time periods and base time periods, such as quarter and year total.

summary tree A tree used to roll up accounts for each type of report in summary ledgers. Summary trees enable you to define trees on trees. In a summary tree, the detail values are really nodes on a detail tree or another summary tree (known as the *basis* tree). A summary tree structure specifies the details on which the summary trees are to be built.

syndicate To distribute a production version of the enterprise catalog to partners.

system function In PeopleSoft Receivables, an activity that defines how the system generates accounting entries for the general ledger.

TableSet A means of sharing similar sets of values in control tables, where the actual data values are different but the structure of the tables is the same.

TableSet sharing Shared data that is stored in many tables that are based on the same TableSets. Tables that use TableSet sharing contain the SETID field as an additional key or unique identifier.

target currency The value of the entry currency or currencies converted to a single currency for budget viewing and inquiry purposes.

template A template is HTML code associated with a web page. It defines the layout of the page and also where to get HTML for each part of the page. In PeopleSoft, you use templates to build a page by combining HTML from a number of sources. For a PeopleSoft portal, all templates must be registered in the portal registry, and each content reference must be assigned a template.

territory In PeopleSoft Sales Incentive Management, hierarchical relationships of business objects, including regions, products, customers, industries, and participants.

TimeSpan A relative period, such as year-to-date or current period, that can be used in various PeopleSoft General Ledger functions and reports when a rolling time frame, rather

	than a specific date, is required. TimeSpans can also be used with flexible formulas in PeopleSoft Projects.
trace usage	In PeopleSoft Manufacturing, enables the control of which components will be traced during the manufacturing process. Serial- and lot-controlled components can be traced. This is maintained in the Item Master record.
transaction allocation	In PeopleSoft Enterprise Incentive Management, the process of identifying the owner of a transaction. When a raw transaction from a batch is allocated to a plan context, the transaction is duplicated in the PeopleSoft Enterprise Incentive Management transaction tables.
transaction state	In PeopleSoft Enterprise Incentive Management, a value assigned by an incentive rule to a transaction. Transaction states enable sections to process only transactions that are at a specific stage in system processing. After being successfully processed, transactions may be promoted to the next transaction state and "picked up" by a different section for further processing.
Translate table	A system edit table that stores codes and translate values for the miscellaneous fields in the database that do not warrant individual edit tables of their own.
tree	The graphical hierarchy in PeopleSoft systems that displays the relationship between all accounting units (for example, corporate divisions, projects, reporting groups, account numbers) and determines roll-up hierarchies.
unclaimed transaction	In PeopleSoft Enterprise Incentive Management, a transaction that is not claimed by a node or participant after the allocation process has completed, usually due to missing or incomplete data. Unclaimed transactions may be manually assigned to the appropriate node or participant by a compensation administrator.
universal navigation header	Every PeopleSoft portal includes the universal navigation header, intended to appear at the top of every page as long as the user is signed on to the portal. In addition to providing access to the standard navigation buttons (like Home, Favorites, and signoff) the universal navigation header can also display a welcome message for each user.
user interaction object	In PeopleSoft Sales Incentive Management, used to define the reporting components and reports that a participant can access in his or her context. All Sales Incentive Management user interface objects and reports are registered as user interaction objects. User interaction objects can be linked to a compensation structure node through a compensation relationship object (individually or as groups).
variable	In PeopleSoft Sales Incentive Management, the intermediate results of calculations. Variables hold the calculation results and are then inputs to other calculations. Variables can be plan variables that persist beyond the run of an engine or local variables that exist only during the processing of a section.
VAT exception	Abbreviation for *value-added tax exception.* A temporary or permanent exemption from paying VAT that is granted to an organization. This terms refers to both VAT exoneration and VAT suspension.
VAT exempt	Abbreviation for *value-added tax exempt.* Describes goods and services that are not subject to VAT. Organizations that supply exempt goods or services are unable to recover the related input VAT. This is also referred to as exempt without recovery.
VAT exoneration	Abbreviation for *value-added tax exoneration.* An organization that has been granted a permanent exemption from paying VAT due to the nature of that organization.
VAT suspension	Abbreviation for *value-added tax suspension.* An organization that has been granted a temporary exemption from paying VAT.
warehouse	A PeopleSoft data warehouse that consists of predefined ETL maps, data warehouse tools, and DataMart definitions.

work order In PeopleSoft Services Procurement, enables an enterprise to create resource-based and deliverable-based transactions that specify the basic terms and conditions for hiring a specific service provider. When a service provider is hired, the service provider logs time or progress against the work order.

worksheet A way of presenting data through a PeopleSoft Business Analysis Modeler interface that enables users to do in-depth analysis using pivoting tables, charts, notes, and history information.

worklist The automated to-do list that PeopleSoft Workflow creates. From the worklist, you can directly access the pages you need to perform the next action, and then return to the worklist for another item.

XML schema An XML definition that standardizes the representation of application messages, component interfaces, or business interlinks.

yield by operation In PeopleSoft Manufacturing, the ability to plan the loss of a manufactured item on an operation-by-operation basis.

zero-rated VAT Abbreviation for *zero-rated value-added tax.* A VAT transaction with a VAT code that has a tax percent of zero. Used to track taxable VAT activity where no actual VAT amount is charged. Organizations that supply zero-rated goods and services can still recover the related input VAT. This is also referred to as exempt with recovery.

Index

E

J

K

L

M

N

Q

R

S

T

U

V

W

X