PeopleSoft®

EnterpriseOne Tools 8.94 PeopleBook: Development Standards for Business Function Programming

November 2004

EnterpriseOne Tools 8.94 PeopleBook: Development Standards for Business Function Programming
SKU E1_TOOLS8.94TDS-B 1104

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.
- Documentation updates and printed documentation.
- Additional resources.
- Typographical conventions and visual cues.
- Comments and suggestions.
- Common elements in PeopleBooks.

Note. PeopleBooks document only page elements, such as fields and check boxes, 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.

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

You should be familiar with navigating the system and adding, updating, and deleting information by using PeopleSoft menus, and pages, forms, or windows. 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 applications. For some applications, additional, essential information describing the setup and design of your system appears in a companion volume of documentation called the application fundamentals PeopleBook. Most PeopleSoft product lines have a version of the application fundamentals PeopleBook. The preface of each PeopleBook identifies the application fundamentals PeopleBooks that are associated with that PeopleBook.

The application fundamentals PeopleBook consists of important topics that apply to many or all PeopleSoft applications across one or more product lines. 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 the appropriate application fundamentals PeopleBooks. They provide the starting points for fundamental implementation tasks.

Documentation Updates and Printed 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

Additional Resources

The following resources are located on the PeopleSoft Customer Connection website:

Resource	Navigation
Application maintenance information	Updates + Fixes
Business process diagrams	Support, Documentation, Business Process Maps
Interactive Services Repository	Interactive Services Repository
Hardware and software requirements	Implement, Optimize + Upgrade, Implementation Guide, Implementation Documentation & Software, Hardware and Software Requirements
Installation guides	Implement, Optimize + Upgrade, Implementation Guide, Implementation Documentation & Software, Installation Guides and Notes
Integration information	Implement, Optimize + Upgrade, Implementation Guide, Implementation Documentation and Software, Pre-built Integrations for PeopleSoft Enterprise and PeopleSoft EnterpriseOne Applications
Minimum technical requirements (MTRs) (EnterpriseOne only)	Implement, Optimize + Upgrade, Implementation Guide, Supported Platforms
PeopleBook documentation updates	Support, Documentation, Documentation Updates
PeopleSoft support policy	Support, Support Policy
Prerelease notes	Support, Documentation, Documentation Updates, Category, Prerelease Notes
Product release roadmap	Support, Roadmaps + Schedules
Release notes	Support, Documentation, Documentation Updates, Category, Release Notes
Release value proposition	Support, Documentation, Documentation Updates, Category, Release Value Proposition
Statement of direction	Support, Documentation, Documentation Updates, Category, Statement of Direction

Resource	Navigation
Troubleshooting information	Support, Troubleshooting
Upgrade documentation	Support, Documentation, Upgrade Documentation and Scripts

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, business function names, event names, system 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.

Typographical Convention or Visual Cue	Description
. . . (ellipses)	Indicate that the preceding item or series can be repeated any number of times in PeopleCode syntax.
{ } (curly braces)	Indicate a choice between two options in PeopleCode syntax. Options are separated by a pipe (|).
[ ] (square brackets)	Indicate optional items in PeopleCode syntax.
& (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.

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.

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 Used in PeopleBooks

Address Book Number Enter a unique number that identifies the master record for the entity. An address book number can be the identifier for a customer, supplier, company, employee, applicant, participant, tenant, location, and so on. Depending on the application, the field on the form might refer to the address book number as the customer number, supplier number, or company number, employee or applicant id, participant number, and so on.

As If Currency Code

Enter the three-character code to specify the currency that you want to use to view transaction amounts. This code allows you to view the transaction amounts as if they were entered in the specified currency rather than the foreign or domestic currency that was used when the transaction was originally entered.

Batch Number

Displays a number that identifies a group of transactions to be processed by the system. On entry forms, you can assign the batch number or the system can assign it through the Next Numbers program (P0002).

Batch Date

Enter the date in which a batch is created. If you leave this field blank, the system supplies the system date as the batch date.

Batch Status

Displays a code from user-defined code (UDC) table 98/IC that indicates the posting status of a batch. Values are:

Blank: Batch is unposted and pending approval.

A: The batch is approved for posting, has no errors and is in balance, but it has not yet been posted.

D: The batch posted successfully.

E: The batch is in error. You must correct the batch before it can post.

P: The system is in the process of posting the batch. The batch is unavailable until the posting process is complete. If errors occur during the post, the batch status changes to E.

U: The batch is temporarily unavailable because someone is working with it, or the batch appears to be in use because a power failure occurred while the batch was open.

Branch/Plant

Enter a code that identifies a separate entity as a warehouse location, job, project, work center, branch, or plant in which distribution and manufacturing activities occur. In some systems, this is called a business unit.

Business Unit

Enter the alphanumeric code that identifies a separate entity within a business for which you want to track costs. In some systems, this is called a branch/plant.

Category Code

Enter the code that represents a specific category code. Category codes are user-defined codes that you customize to handle the tracking and reporting requirements of your organization.

Company

Enter a code that identifies a specific organization, fund, or other reporting entity. The company code must already exist in the F0010 table and must identify a reporting entity that has a complete balance sheet.

Currency Code

Enter the three-character code that represents the currency of the transaction. PeopleSoft EnterpriseOne provides currency codes that are recognized by the International Organization for Standardization (ISO). The system stores currency codes in the F0013 table.

Document Company

Enter the company number associated with the document. This number, used in conjunction with the document number, document type, and general ledger date, uniquely identifies an original document.

If you assign next numbers by company and fiscal year, the system uses the document company to retrieve the correct next number for that company.

If two or more original documents have the same document number and document type, you can use the document company to display the document that you want.

Document Number — Displays a number that identifies the original document, which can be a voucher, invoice, journal entry, or time sheet, and so on. On entry forms, you can assign the original document number or the system can assign it through the Next Numbers program.

Document Type — Enter the two-character UDC, from UDC table 00/DT, that identifies the origin and purpose of the transaction, such as a voucher, invoice, journal entry, or time sheet. PeopleSoft EnterpriseOne reserves these prefixes for the document types indicated:

P: Accounts payable documents.

R: Accounts receivable documents.

T: Time and pay documents.

I: Inventory documents.

O: Purchase order documents.

S: Sales order documents.

Effective Date — Enter the date on which an address, item, transaction, or record becomes active. The meaning of this field differs, depending on the program. For example, the effective date can represent any of these dates:

- The date on which a change of address becomes effective.
- The date on which a lease becomes effective
- The date on which a price becomes effective.
- The date on which the currency exchange rate becomes effective.
- The date on which a tax rate becomes effective.

Fiscal Period and **Fiscal Year** — Enter a number that identifies the general ledger period and year. For many programs, you can leave these fields blank to use the current fiscal period and year defined in the Company Names & Number program (P0010)

G/L Date (general ledger date) — Enter the date that identifies the financial period to which a transaction will be posted. The system compares the date that you enter on the transaction to the fiscal date pattern assigned to the company to retrieve the appropriate fiscal period number and year, as well as to perform date validations.

PeopleSoft EnterpriseOne Development Standards for Business Function Programming Preface

This preface provides a general overview of the contents discussed in the PeopleSoft EnterpriseOne 8.94 PeopleBook: Development Standards for Business Function Programming.

Development Standards for Business Function Programming

This PeopleBook covers a wide range of considerations and rules for developing business functions for PeopleSoft EnterpriseOne, including:

- Program Flow
- Program Flow
- Readability
- Declaring and Initializing Variables and Data Structures
- General Coding Guidelines
- Portability
- EnterpriseOne Defined Structures
- Error Messages
- Data Dictionary Triggers
- Unicode Compliance Standards
- Standard Header and Source Files

CHAPTER 1

Getting Started with Development Standards for Business Function Programming

This chapter provides overviews of development standards for business function programming and program flow.

Understanding Development Standards for Business Function Programming

A business function is an integral part of the PeopleSoft EnterpriseOne tool set. A business function allows application developers to attach custom functionality to application and batch processing events. Business function are programmed in C code.

A standard method for creating C code that performs a specific action does not exist. C code can be as unique as the individual who programmed and compiled the code. This guide provides standards for coding business functions that are efficient, readable, and easy to maintain.

A business function is also referred to as an EnterpriseOne business function because it is partially created using the EnterpriseOne software tools. A business function is checked in and checked out so that it is available to other applications and business functions.

Understanding Program Flow

The program flow of a C function should be from the top down, modularizing the code segments. For readability and maintenance purposes, divide code into logical units.

Internal functions are created for modularity of a common routine that is called by the business function.

Other Sources of Information

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

CHAPTER 2

Understanding Naming Conventions

Standardized naming conventions ensure a consistent approach to identifying function objects and function sections.

This chapter provides overviews of source and header file names, function names, variable names, and business function data structure names.

Source and Header File Names

Source and header file names can be a maximum of 8 characters and should be formatted as *bxxyyyy*, where:

- *b* = Source or header file
- *xx* (second two digits) = The system code, such as:
  - 01 = Address Book
  - 04 = Accounts Payable
- *yyyyy* (the last five digits) = A sequential number for the system code, such as:
  - 00001 = The first source or header file for the system code
  - 00002 = The second source or header file for the system code

Both the C source and the accompanying header file should have the same name.

The following table shows examples of this naming convention:

System	System Code	Source Number	Source File	Header File
Address Book	01	10	b0100010.c	b0100010.h
Accounts Receivable	04	58	b0400058.c	b0400058.h
General Ledger	09	2457	b0902457.c	b0902457.h

Function Names

An internal function can be a maximum of 42 characters and should be formatted as *Ixxxxxx_a*, where:

- *I* = An internal function
- *xxxxxx* = The source file name

- *a* = The function description

  Function descriptions can be up to 32 characters in length. Be as descriptive as possible and capitalize the first letter of each word, such as ValidateTransactionCurrencyCode. When possible use the major table name or purpose of the function.

An example of a Function Name is I4100040_CompareDate

Note. Do not use an underscore after I.

Variable Names

Variables are storage places in a program and can contain numbers and strings. Variables are stored in the computer's memory. Variables are used with keywords and functions, such as char and MATH_NUMERIC, and must be declared at the beginning of the program.

A variable name can be up to 32 characters in length. Be as descriptive as possible and capitalize the first letter of each word.

You must use Hungarian prefix notation for all variable names, as shown in the following table:

Prefix	Description
c	JCHAR
sz	NULL-terminated JCHAR string
z	ZCHAR
zz	NULL-terminated ZCHAR string
n	short
l	long
b	Boolean
mn	MATH_NUMERIC
jd	JDEDATE
lp	long pointer
i	integer
by	byte
ul	unsigned long (identifier)
us	unsigned Short
ds	data structures
h	handle

Prefix	Description
e	enumerated types
id	id long integer, JDE data type for returns
ut	JDEUTIME
sz	VARCHAR

Example: Hungarian Notation for Variable Names

The following variable names use Hungarian notation:

JCHAR	cPaymentRecieved;
JCHAR	szCompanyNumber = _J(00000);
short	nLoopCounter;
long int	lTaxConstant;
BOOL	bIsDateValid;
MATH_NUMERIC	mnAddressNumber;
JDEDATE	jdGLDate;
LPMATH_NUMERIC	lpAddressNumber;
int	iCounter;
byte	byOffsetValue;
unsigned long	ulFunctionStatus;
D0500575A	dsInputParameters;
JDEDB_RESULT	idJDEDBResult;

Business Function Data Structure Names

The data structure for business function event rules and business functions should be formatted as *DxxyyyyA*, where:

- *D* = Data structure
- *xx* (second two digits) = The system code, such as
  - 01 = Address Book
  - 02 = Accounts Payable

- *yyyy* = A next number (the numbering assignments follow current procedures in the respective application groups)
- *A* = An alphabetical character (such as A, B, C and so on) placed at the end of the data structure name to indicate that a function has multiple data structures

  Even if a function has only one data structure, you should include the A in the name.

An example of a Business Function Data Structure Name is D050575A.

CHAPTER 3

Understanding Readability

Readable code is easier to debug and maintain. The following section presents guidelines for making your code more readable by maintaining the change log, inserting comments, indenting code, and formatting compound statements.

This chapter provides overviews of maintaining the source and header code change log, indenting code, and formatting compound statements.

Maintaining the Source and Header Code Change Log

You must note any code changes that you make to the standard source and header for a business function. Include the following information:

- *SAR* - the SAR number
- *Date* - the date of the change
- *Initials* - the programmer's initials
- *Comment* - the reason for the change

Inserting Comments

Insert comments that describe the purpose of the business function and your intended approach. Using comments will make future maintenance and enhancement of the function easier.

Use the following checklist for inserting comments:

- Always use the /*comment */ style. The use of // comments is not portable.
- Precede and align comments with the statements they describe.
- Comments should never be more that 80 characters wide.

Example: Inserting Comments

The following example shows the correct way to insert block and inline comments into code:

```
/*-------------------------------------------------------
 * Comment blocks need to have separating lines between
 * the text description. The separator can be a
 * dash '-' or an asterisk '*'
```

```
*-------------------------------------------------------*/
if ( statement )
{
   statements
} /* inline comments indicate the meaning of one statement */
/*-------------------------------------------------------
* Comments should be used in all segments of the source
* code. The original programmer may not be the programmer
* maintaining the code in the future which makes this a
* crucial step in the development process.
*-------------------------------------------------------*/
/*********************************************************
* Function Clean Up
*********************************************************/
```

Indenting Code

Any statements executed inside a block of code should be indented within that block of code. Standard indentation is three spaces.

Note. Set up the environment for the editor you are using to set tab stops at 3 and turn the tab character off. Then, each time you press the Tab key, three spaces are inserted rather than the tab character. Turn on auto-indentation.

Example: Indenting Code

The following is the standard method to indent code:

```
function block
{
   if ( nJDEDBReturn == JDEDB_PASSED )
   {
      CallSomeFunction( nParameter1, szParameter2 );
      CallAnotherFunction( lSomeNumber );
      while( FunctionWithBooleanReturn() )
      {
         CallYetAnotherFunction( cStatusCode );
      }
   }
}
```

Formatting Compound Statements

Compound statements are statements followed by one or more statements enclosed with braces. A function block is an obvious example of a compound statement. Control statements (while, for) and selection statements (if, switch) are also examples of compound statements.

Omitting braces is a common C coding practice when only one statement follows a control or selection statement. However, you must use braces for all compound statements for the following reasons:

- The absence of braces can cause errors.
- Braces ensure that all compound statements are treated the same way.
- In the case of nested compound statements, the use of braces clarifies the statements that belong to a particular code block.
- Braces make subsequent modifications easier.

Refer to the following guidelines when formatting compound statements:

- Always have one statement per line within a compound statement.
- Always use braces to contain the statements that follow a control statement or a selection statement.
- Braces should be aligned with the initial control or selection statement.
- Logical expressions evaluated within a control or selection statement should be broken up across multiple lines if they do not fit on one line. When breaking up multiple logical expressions, do not begin a new line with the logical operator; the logical operator must remain on the preceding line.
- When evaluating multiple logical expressions, use parentheses to explicitly indicate precedence.
- Never declare variables within a compound statement, except function blocks.
- Use braces for all compound statements.
- Place each opening or closing brace, { or }, on a separate line.

Example: Formatting Compound Statements

The following example shows how to format compound statements for ease of use and to prevent mistakes:

```
/*
 * Do the Issues Edit Line if the process edits is either
 * blank or set to SKIP_COMPLETIONS. The process edits is
 * set to SKIP_COMPLETIONS if Hours and Quantities is in
 * interactive mode and Completions is Blind in P31123.
 */
if ((dsWorkCache.PO_cIssuesBlindExecution == _J('1')) &&
   ((dsCache.cPayPointCode == _J('M'))      ||
    (dsCache.cPayPointCode == _J('B')))       &&
   (lpDS->cProcessEdits != ONLY_COMPLETIONS))
{
   /* Process the Pay Point line for Material Issues */
   idReturnCode = I3101060_BlindIssuesEditLine(&dsInternal,
                                               &dsCache,
                                               &dsWorkCache);
}
```

Example: Using Braces to Clarify Flow

The following example shows the use of braces to clarify the flow and prevent mistakes:

```
if(idJDBReturn != JDEDB_PASSED)
```

```
{
   /* If not add mode, record must exist */
   if ((lpdsInternal->cActionCode != ADD_MODE) &&
       (lpdsInternal->cActionCode != ATTACH_MODE))
   {
      /* Issue Error 0002 - Work Order number invalid */
      jdeStrncpy((JCHAR*)(lpdsInternal->szErrorMessageID),
                 (const JCHAR*)_J(0002),
                 DIM(lpdsInternal->szErrorMessageID)-1);
      lpdsInternal->idFieldID = IDERRmnOrderNumber_15;
      idReturnCode = ER_ERROR;
   }
}
else
{
   /* If in add mode and the record exists, issue error and exit */
   if (lpdsInternal->cActionCode == ADD_MODE)
   {
      /* Issue Error 0002 - Work Order number invalid */
      jdeStrncpy((JCHAR*)(lpdsInternal->szErrorMessageID),
                 (const JCHAR*)_J(0002),
                 DIM(lpdsInternal->szErrorMessageID)-1);
      lpdsInternal->idFieldID = IDERRmnOrderNumber_15;
      idReturnCode = ER_ERROR;
   }
   else
   {
      /*
       * Set flag used in determining if the F4801 record should be sent
       * in to the modules
       */
       lpdsInternal->cF4801Retrieved = _J('1');
   }
}
```

Example: Using Braces for Ease in Subsequent Modifications

The use of braces prevents mistakes when the code is later modified. Consider the following example. The original code contains a test to see if the number of lines is less than a predefined limit. As intended, the return value is assigned a certain value if the number of lines is greater than the maximum. Later, someone decides that an error message should be issued in addition to assigning a certain return value. The intent is for both statements to be executed only if the number of lines is greater than the maximum. Instead, idReturn will be set to ER_ERROR regardless of the value of nLines. If braces were used originally, this mistake would have been avoided.

```
ORIGINAL

if (nLines > MAX_LINES)
   idReturn = ER_ERROR;

MODIFIED
```

```
if (nLines > MAX_LINES)
   jdeErrorSet (lpBhvrCom, lpVoid,
               (ID) 0, _J(4353), (LPVOID) NULL):
   idReturn = ER_ERROR;

STANDARD ORIGINAL

if (nLines > MAX_LINES)
{
   idReturn = ER_ERROR;
}

STANDARD MODIFIED

if (nLines > MAX_LINES)
{
   jdeErrorSet (lpBhvrCom, lpVoid,
               (ID) 0, _J(4363), (LPVOID) NULL);
   idReturn = ER_ERROR;
}
```

Example: Handling Multiple Logical Expressions

The following example shows how to handle multiple logical expressions:

```
while ( (lWorkArray[elWorkX] < lWorkArray[elWorkMAX]) &&
        (lWorkArray[elWorkX] < lWorkArray[elWorkCDAYS]) &&
        (idReturnCode == ER_SUCCESS))


{
   statements
}
```

CHAPTER 4

Understanding Declaring and Initializing Variables and Data Structures

Variables and data structures must be defined and initialized before they can be used to store data. This chapter describes how to declare and initialize them. It includes topics on using define statements, using typedef, creating function prototypes, initializing variables, initializing data structures, and using standard variables.

This chapter provides overviews of using define statements, using typedef statements, creating function prototypes, initializing variables, initializing data structures, and using standard variables.

Using Define Statements

A define statement is a directive that sets up constants at the beginning of the program. A define statement always begins with a pound sign (#). All business functions include the system header file: jde.h. System-wide define statements are included in the system header file.

If you need define statements for a specific function, include the define statement in uppercase letters within the source file for the function whenever possible. The statement should directly follow the header file inclusion statement.

Usually, you should place define statements in the source file, not the header file. When placed in the header file, you can redefine the same constant with different values, causing unexpected results. However, rare cases exist when it is necessary to place a define statement in the function header file. In these cases, precede the definition name with the business function name to ensure uniqueness.

Example: #define in Source File

The following example includes define statements within a business function source file:

```
/*************************************************************
* Notes
*************************************************************/


#include <bxxxxxxx.h>


/*************************************************************
 * Global Definitions
 *************************************************************/
#define CACHE_GET            '1'
#define CACHE_ADD            '2'
```

```
#define CACHE_UPDATE         '3'
#define CACHE_DELETE         '4'
```

Example: #define in Header File

The following example includes define statements within a business function header:

```
/*********************************************************
 * External Business Function Header Inclusions
 *********************************************************/

#include <bxxxxxxx.h>

/*********************************************************
 * Global definitions
 *********************************************************/
#define BXXXXXXX_CACHE_GET         '1'
#define BXXXXXXX_CACHE_ADD         '2'
#define BXXXXXXX_CACHE_UPDATE      '3'
#define BXXXXXXX_CACHE_DELETE      '4'
```

Using Typedef Statements

When using typedef statements, always name the object of the typedef statement using a descriptive, uppercase format. If you are using a typedef statement for data structures, remember to include the name of the business function in the name of the typedef to make it unique. See the following example for using a typedef statement for a data structure.

Example: Using Typedef for a User-Defined Data Structure

The following is an example of a user-defined data structure:

```
/************************************************************
 * Structure Definitions
 ************************************************************/

typedef struct
{
   HUSER         hUser;           /** User handle **/
   HREQUEST      hRequestF0901;   /** File Pointer to the
                                    * Account Master **/
   DSD0051       dsData;          /** X0051 - F0902 Retrieval **/
   int           iFromYear;       /** Internal Variables **/
   BOOL          bProcessed;
   MATH_NUMERIC  mnCalculatedAmount;
   JCHAR         szSummaryJob[13];
   JDEDATE       jdStartPeriodDate;
} DSX51013_INFO, *LPDSX51013_INFO;
```

Creating Function Prototypes

Refer to the following guidelines when defining function prototypes:

- Always place function prototypes in the header file of the business function in the appropriate prototype section.
- Include function definitions in the source file of the business function, preceded by a function header.
- Ensure that function names follow the naming convention defined in this guide.
- Ensure that variable names in the parameter list follow the naming convention defined in this guide.
- List the variable names of the parameters along with the data types in the function prototype.
- List one parameter per line so that the parameters are aligned in a single column.
- Do not allow the parameter list to extend beyond 80 characters in the function definition. If the parameter list must be broken up, the data type and variable name must stay together. Align multiple-line parameter lists with the first parameter.
- Include a return type for every function. If a function does not return a value, use the keyword void as the return type.
- Use the keyword "void" in place of the parameter list if nothing is passed to the function.

Example: Creating a Business Function Prototype

The following is an example of a standard business function prototype:

```
/*****************************************************************
 * Business Function: BusinessFunctionName
 *
 *    Description: Business Function Name
 *
 *     Parameters:
 *      LPBHVRCOM  lpBhvrCom Business Function Communications
 *      LPVOID   lpVoid  Void Parameter - DO NOT USE!
 *    LPDSD51013 lpDS   Parameter Data Structure Pointer
 *
 *****************************************************************/

JDEBFRTN (ID) JDEBFWINAPI BusinessFunctionName (LPBHVRCOM  lpBhvrCom,
                                                LPVOID     lpVoid,
                                                LPDSXXXXXX lpDS)
```

Example: Creating an Internal Function Prototype

The following is an example of a standard internal function prototype:

```
Type XXXXXXXX_AAAAAAAA( parameter list ... );

type     : Function return value
XXXXXXXX : Unique source file name
AAAAAAAA : Function Name
```

Example: Creating an External Business Function Definition

The following is an example of a standard external business function definition:

```
/*
 *  see sample source for standard business function heading
 */
JDEBFRTN (ID) JDEBFWINAPI GetAddressBookDescription(LPBHVRCOM  lpBhvrCom,
                                                    LPVOID     lpVoid,
                                                    LPDSNNNNNN lpDS)
{
   ID idReturn = ER_SUCCESS;
   /*------------------------------------------------------------
    * business function code
    */
   return idReturn;
}
```

Example: Creating an Internal Function Definition

The following is an example of a standard internal function definition:

```
/*------------------------------------------------------------------------
 *  see sample source for standard function header
 */
void I4100040_GetSupervisorManagerDefault( LPBHVRCOM lpBhvrCom,
                                           LPSTR lpszCostCenterIn,
                                           LPSTR lpszManagerOut,
                                           LPSTR lpszSupervisorOut )
/*------------------------------------------------------------------------
 * Note: b4100040 is the source file name
 */
{
   /*
    * internal function code
    */
}
```

See Also

Chapter 11, “Understanding Standard Header and Source Files,” Business Function Prototypes, page 58

Initializing Variables

Variables store information in memory that is used by the program. Variables can store strings of text and numbers.

When you declare a variable, you should also initialize it. Two types of variable initialization exist: explicit and implicit. Variables are explicitly initialized if they are assigned a value in the declaration statement. Implicit initialization occurs when variables are assigned a value during processing.

The following information covers standards for declaring and initializing variables in business functions and includes an example of standard formats.

Use the following guidelines when declaring and initializing variables:

- Declare variables using the following format:

```
datatype variable name = initial value; /* descriptive comment*/
```

- Declare all variables used within business functions and internal functions at the beginning of the function. Although C allows you to declare variables within compound statement blocks, this standard requires all variables used within a function to be declared at the beginning of the function block.
- Declare only one variable per line, even if multiple variables of the same type exist. Indent each line three spaces and left align the data type of each declaration with all other variable declarations. Align the first character of each variable name (variable name in the format example above) with variable names in all other declarations.
- Use the naming conventions set forth in this guide. When initializing variables, the initial value is optional depending on the data type of the variable. Generally, all variables should be explicitly initialized in their declaration.
- The descriptive comment is optional. In most cases, variable names are descriptive enough to indicate the use of the variable. However, provide a comment if further description is appropriate or if an initial value is unusual.
- Left align all comments.
- Data structures should be initialized to zero using memset immediately after the declaration section.
- Some Application Program Interfaces (APIs), such as the JDB ODBC API, provide initialization routines. In this case, the variables intended for use with the API should be initialized with the API routines.
- Always initialize pointers to NULL and include an appropriate type call at the declaration line.
- Initialize all variables, except data structures, in the declaration.
- Initialize all declared data structures, MATH_NUMERIC, and JDEDATE to NULL.
- Ensure that the byte size of the variable matches the size of the data structure you want to store.

Example: Initializing Variables

The following example shows how to initialize variables:

```
JDEBFRTN (ID) JDEBFWINAPI F0902GLDateSensitiveRetrieval
                          (LPBHVRCOM   lpBhvrCom,
                           LPVOID     lpVoid,
                           LPDSD0051   lpDS)
/************************************************************
 * Variable declarations
 ************************************************************/
   ID             idReturn      = ER_SUCCESS;
   JDEDB_RESULT   eJDEDBResult  = JDEDB_PASSED;
   long           lDateDiff     = 0L;
   BOOL           bAddF0911Flag = TRUE;
   MATH_NUMERIC   mnPeriod      = {0};

/************************************************************
```

```
 * Declare structures
 ************************************************************/
   HUSER          hUser         = (HUSER) NULL;
   HREQUEST       hRequestF0901 = (HREQUEST) NULL;
   DSD5100016     dsDate        = {0};
   JDEDATE        jdMidDate     = {0};

/************************************************************
 * Pointers
 ************************************************************/
   LPX0051_DSTABLES lpdsTables = (LPX0051_DSTABLES) 0L;

/************************************************************
 * Check for NULL pointers
 ************************************************************/
   if ((lpBhvrCom == (LPBHVRCOM) NULL) ||
       (lpVoid    == (LPVOID)    NULL) ||
       (lpDS      == (LPDSD0051)   NULL))
   {
      jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0,
                   _J(4363), (LPVOID) NULL);
      return ER_ERROR;
   }

/************************************************************
 * Main Processing
 ************************************************************/
   eJDEDBResult = JDB_InitBhvr ((void*)lpBhvrCom,
                                &hUser,
                                (JCHAR *) NULL,
                                JDEDB_COMMIT_AUTO);

 memcopy ((void*)) &dsDate.jdPeriodEndDate,
          (const void*) &lpDS->jdGLDate, sizeof(JDEDATE));
```

Initializing Data Structures

When writing to the table, the table recognizes the following default values:

- Space-NULL if string is blank
- 0 value if math numeric is 0
- 0 JDEDATE if date is blank
- Space if character is blank

Always memset to NULL on the data structure that is passed to another business function to update a table or fetch a table.

Example: Using Memset to Reset the Data Structure to Null

The following example resets the data structure to NULL when initializing the data structure:

```
bOpenTable = B5100001_F5108SetUp( lpBhvrCom, lpVoid,
                                  lphUser, &hRequestF5108);

if ( bOpenTable )
{
   memset( (void *)(&dsF5108Key), 0x00, sizeof(KEY1_F5108) );
   jdeStrcpy( (JCHAR*) dsF5108Key.mdmcu,
              (const JCHAR*) lpDS->szBusinessUnit );
   memset( (void *)(&dsF5108), 0x00, sizeof(F5108) );

   jdeStrcpy( (JCHAR*) dsF5108.mdmcu,
            (const JCHAR*) lpDS->szBusinessUnit );
   MathCopy(&dsF5108.mdbsct, &mnCentury);
   MathCopy(&dsF5108.mdbsfy, &mnYear);
   MathCopy(&dsF5108.mdbtct, &mnCentury);
   MathCopy(&dsF5108.mdbtfy, &mnYear);
   eJDEDBResult = JDB_InsertTable( hRequestF5108,
                                   ID_F5108,
                                   (ID)(0),
                                   (void *) (&dsF5108) );
}
```

Using Standard Variables

This section presents the requirements for standard variables, including flag variables, input and output parameters, and fetch variables.

Flag Variables

When creating flag variables, use the following guidelines:

- Any true-or-false flag used must be a Boolean type (BOOL).
- Name the flag variable to answer a question of TRUE or FALSE.

Examples of flag variables are listed below, with a brief description of how each is used:

Flag Variable	Description
bIsMemoryAllocated	Apply to memory allocation
bIsLinkListEmpty	Link List

Input and Output Parameters

Business functions frequently return error codes and pointers. The input and output parameters in the business function data structure should be named as follows:

Input and Output Parameter	Description
cReturnPointer	When allocating memory and returning GENLNG.
cErrorCode	Based on cCallType, cErrorCode returns a 1 when it fails or a 0 when it succeeds.
cSuppressErrorMessage	If the value is 1, do not display error message using jdeErrorSet(...). If the value is 0, display the error.
szErrorMessageId	If an error occurs, return an error message ID (value). Otherwise, return four spaces.

Fetch Variables

Use fetch variables to retrieve and return specific information, such as a result; to define the table ID; and to specify the number of keys to use in a fetch.

Fetch Variable	Description
idJDEDBResult	APIs or PeopleSoft EnterpriseOne functions, such as JDEDB_RESULT
idReturnValue	Business function return value, such as ER_WARNING or ER_ERROR
idTableXXXXID	Where XXXX is the table name, such as F4101 and F41021, the variable used to define the Table ID.
idIndexXXXXID	Where XXXX is the table name, such as F4101 or F41021, the variable used to define the Index ID of a table.
usXXXXNumColToFetch	Where XXXX is the table name, such as F4101 and F41021, the number of the column to fetch. *Do not* put the literal value in the API functions as the parameter.
usXXXXNumOfKeys	Where XXXX is the table name, such as F4101 and F41021, the number of keys to use in the fetch.

Example: Using Standard Variables

The following example illustrates the use of standard variables:

```
/************************************************************
  * Variable declarations
  ************************************************************/
 ID      idJDEDBResult  = JDEDB_PASSED;
 ID      idTableF0901   = ID_F0901;
 ID      idIndexF0901   = ID_F0901_ACCOUNT_ID;
```

```
ID      idFetchCol[]   = { ID_CO, ID_AID, ID_MCU, ID_OBJ,
                           ID_SUB, ID_LDA, ID_CCT };
ushort    usNumColToFetch = 7;
ushort    usNumOfKeys     = 1;

/************************************************************
 * Structure declarations
 ************************************************************/
KEY3_F0901     dsF0901Key;
DSX51013_F0901 dsF0901;

/************************************************************
 * Main Processing
 ************************************************************/
/** Open the table, if it is not open **/
  if ((*lpdsInfo->lphRequestF0901) == (HREQUEST) NULL)
  {
     if ( (*lpdsInfo->lphUser) == (HUSER) 0L )
     {
        idJDEDBResult = JDB_InitBhvr ((void*)lpBhvrCom,
                                      &lpdsInfo->lphUser,
                                      (JCHAR *) NULL,
                                      JDEDB_COMMIT_AUTO);
     }
     if (idJDEDBResult == JDEDB_PASSED)
     {
        idJDEDBResult = JDB_OpenTable( (*lpdsInfo->lphUser),
                                        idTableF0901,
                                        idIndexF0901,
                                        (LPID)(idFetchCol),
                                        (ushort)(usNumColFetch),
                                        (JCHAR *) NULL,
                                        &lpdsInfo->hRequestF0901 );
     }
  }
/** Retrieve Account Master - AID only sent **/
if (idJDEDBResult == JDEDB_PASSED)
{
   /** Set Key and Fetch Record **/
   memset( (void *)(&dsF0901Key),
           (int) _J('\0'), sizeof(KEY3_F0901) );
   jdeStrcpy ((char *) dsF0901Key.gmaid,
              (const JCHAR*) lpDS->szAccountID );
   idJDEDBResult = JDB_FetchKeyed ( lpdsInfo->hRequestF0901,
                                    idIndexF0901,
                                    (void *)(&dsF0901Key),
                                    (short)(1),
                                    (void *)(&dsF0901),
                                    (int)(FALSE) );
   /** Check for F0901 Record **/
```

```
   if (eJDEDBResult == JDEDB_PASSED)
   {
      statement
   }
}
```

CHAPTER 5

Understanding General Coding Guidelines

This chapter discusses how to:

- Use function calls.
- Pass pointers between business functions.
- Allocate and release memory.
- Use hRequest and hUser.
- Typecast
- Comparison test.
- Copy strings with jdeStrcpy.
- Use the function clean up area.
- Insert function exit points.
- Terminate a function.

Using Function Calls

Reuse of existing functions through a function call prevents duplicate code. Refer to the following guidelines when using function calls:

- Always put a space between each parameter.
- If the function has a return value, always check the return of the function for errors or a valid value.
- Use jdeCallObject to call another business function.
- When calling functions with long parameter lists, the function call should not be wider than 80 characters.

  Break the parameter list into one or more lines, aligning the first parameter of proceeding lines with the first parameter in the parameter list.
- Make sure the data types of the parameters match the function prototype.

  When intentionally passing variables with data types that do not match the prototype, explicitly cast the parameters to the correct data type.

Calling an External Business Function

Use jdeCallObject to call an external business function defined in the Object Management Workbench. Include the header file for the external business function that contains the prototype and data structure definition. It is good practice to check the value of the return code.

Example: Calling an External Business Function

The following example calls an external business function:

```
/*------------------------------------------------------------
 *
 * Retrieve account master information
 *
 *------------------------------------------------------------*/
   idReturnCode = jdeCallObject(_J("ValidateAccountNumber),
                                NULL,
                                lpBhvrCom,
                                lpVoid,
                                (void*) &dsValidateAccount,
                                (CALLMAP*) NULL,
                                (int) 0,
                                (JCHAR*) NULL,
                                (JCHAR*) NULL,
                                (int) 0 );
   if ( idReturnCode == ER_SUCCESS )
   {
      statement;
   }
```

Calling an Internal Business Function

You can access internal business functions (internal C functions) within the same source file.

You may create modular subroutines that can be accessed from multiple source files. Use `CALLIBF(fcn(parm1,parm2))` and`CALLIBFRET(ret,fcn(parm1,parm2))` to access internal business functions within a different source file but within the same DLL. Use CALLIBF to call an internal business function with no return value. Use CALLIBFRET to call an internal business function with a return value. Both CALLIBF and CALLIFBRET can call internal business functions with any type or number of parameters.

CALLIBF and CALLIBFRET can only call internal functions within the same business function DLL. They cannot call functions in other business function DLLs. For example, if the internal function intFcn123() is in B550001.C, which is in the CALLBSFN.DLL, you cannot called it with CALLIBF or CALLIBFRET from a business function in CDIST.DLL.

To use CALLIBF or CALLIBFRET for an internal business function, the business function must have its prototype in the business function header. If you do not want other modules calling your internal business function, place the prototype in the C file, not the header file.

Calling internal business functions has several advantages over external business functions. First, they do not have the jdeCallObject performance overhead of checking OCM mapping and possibly executing the function remotely. A called function always executes in the same process from where it was called. Second, the parameters are not restricted to EnterpriseOne data dictionary data types. Any valid C data type, including pointers, may be passed in and out of internal functions.

Example: Calling an Internal Business Function with No Return Value

The following example calls an internal business function that has no return value.

b550001.h

The following is an example of b550001.h:

```
/* normal business function header pieces */
...
/* The internal business function prototype must be in the header for other
   modules to call it */
   void i550001(int *a, int b);
```

b550001.c

The following is an example of b550001.c:

```
/* normal business function code pieces */
#include <b550001.h>
JDEBFRTN(ID) JDEBFWINAPI TestBSFN(LPBHVRCOM  lpVhvrCom,
                                  LPVOID   lpVoid,
                                  LPDSB550001 lpDS)
{
...
}
void i550001(int *a, int b)
{
   *a = *a + b;
   return;
}
```

b550002.c

The following is an example of b550002.c:

```
/* normal business function code pieces */
#include <b550002.h>
#include <b550001.h>

JDEBFRTN(ID) JDEBFWINAPI TestBSFN(LPBHVRCOM  lpBhvrCom,
                                  LPVOID   lpVoid,
                                  LPDSB550001 lpDS)
{
   int total = 3;
   int adder = 7;

   CALLIBF(i550001(&total,adder));
}
```

Example: Calling an Internal Business Function with a Return Value

The following example calls an internal business function that has a return value.

b550001.h

The following is an example of b550001.h:

```
/* normal business function header pieces */
...
/* The internal business function prototype must be in the header for
other modules to call it */

   int i550001(int a, int b);
```

b550001.c

The following is an example of b550001.c:

```
/* normal business function code pieces */
#include <b550001.h>

JDEBFRTN(ID) JDEBFWINAPI TestBSFN(LPBHVRCOM   lpBhvrCom,
                                  LPVOID      lpVoid,
                                  LPDSB550001 lpDS)
{
...
}
int i550001(int a, int b)
{
   a = a + b;
   return;
}
```

b550002.c

The following is an example form b550002.c:

```
/* normal business function code pieces */
#include <b550002.h>
#include <b550001.h>

JDEBFRTN(ID) JDEBFWINAPI TestBSFN(LPBHVRCOM   lpBhvrCom,
                                  LPVOID      lpVoid,
                                  LPDSB550001 lpDS)
{
   int total  = 0;
   int adder1 = 6;
   int adder2 = 7;
   CALLIBFRET(total,i550001(adder1,adder2));
}
```

Passing Pointers between Business Functions

Never pass pointers directly in or out of business functions. A pointer memory address should not be greater than 32 bits. If you pass a pointer address that exceeds 32 bits across the platform to a client that supports just 32 bits, the significant digit might be truncated and invalidate the address.

The correct way to share pointers between business functions is to store the address in an array. This array is located on the server platform specified in the Object Configuration Manager (OCM). The array allows up to 100 memory locations to be allocated and stored, and it is maintained by EnterpriseOne tools. The index to a position in the array is a long integer type or ID. Use the GENLNG data dictionary object in your business function data structure to pass this index in or out of the business function.

Storing an Address in an Array

Use jdeStoreDataPtr to store an allocated memory pointer in an array for later retrieval. The index to the position in the array is returned. This index should be passed out through the business function data structure (lpDS).

Example: Storing an Address in an Array

The following example illustrates how to store an address in an array:

```
If (lpDS->cReturnF4301PtrFlag == _J('1'))
{
   lpDS->idF4301RowPtr = jdeStoreDataPtr(hUser,(void *)lpdsF4301);
}
```

Retrieving an Address from an Array

Use jdeRetrieveDataPtr to retrieve an address outside the current business function. The index to the position in the array should be passed in through the business function data structure (lpDS). When you use jdeRetrieveDataPtr, the address remains in the array and can be retrieved again later.

Example: Retrieving an Address from an Array

The following example retrieves an address from an array:

```
lpdsF43199 = (LPF43199) jdeRetrieveDataPtr
              (hUser, lpDS->idF43199Pointer);
```

Removing an Address from an Array

Use jdeRemoveDataPtr to remove the address from the array cell and release the array cell. The index to the position in the array should be passed in through the business function data structure (lpDS). A corresponding call to jdeRemoveDataPtr must exist for every jdeStoreDataPtr. If you use jdeAlloc to allocate memory, use jdeFree to free the memory.

Example: Removing an Address from an Array

The following example removes an address from an array:

```
if (lpDS->idGenericLong != (ID) 0)
{
   lpGenericPtr = (void *)jdeRemoveDataPtr(hUser,lpDS->idGenericLong);
   if (lpGenericPtr != (void *) NULL)
   {
      jdeFree((void *)lpGenericPtr);
      lpDS->idGenericLong = (ID) 0;
      lpGenericPtr = (void *) NULL;
   }
```

```
}
```

Allocating and Releasing Memory

Use jdeAlloc to allocate memory. Because jdeAlloc affects performance, use it sparingly.

Use jdeFree to release memory within a business function. For every jdeAlloc, a jdeFree should exist to release the memory.

Note. Use the business function Free Ptr To Data Structure, B4000640, to release memory through event rule logic.

Example: Allocating and Releasing Memory within a Business Function

The following example uses jdeAlloc to allocate memory, and then, in the function cleanup section, jdeFree to release memory:

```
statement
lpdsF4301 = (LPF4301)jdeAlloc( COMMON_POOL,sizeof(F4301),MEM_ZEROINIT ) ;
statement

/************************************************************
 * Function Clean Up Section
 ************************************************************/
if (lpdsF4301 != (LPF4301) NULL)
{
   jdeFree( lpdsF4301 );
}
```

Using hRequest and hUser

Some API calls require hUser and hRequest. To get the hUser, use JDBInitBhvr. To get the hRequest, use JDBOpenTable. Initialize hUser and hRequest to NULL in the variable declaration line. All hRequest and hUser declarations should have JDB_CloseTable( ) and JDB_FreeBhvr( ) in the function cleanup section.

Typecasting

Typecasting is also known as type conversion. Use typecasting when the function requires a certain type of value, when defining function parameters, and when allocating memory with jdeAlloc().

Note. This standard is for all function calls as well as function prototypes.

Comparison Testing

Always use explicit tests for comparisons. Do not embed assignments in comparison tests. Assign a value or result to a variable and use the variable in the comparison test.

Always test floating point variables using <= or >=. Do not use == or != since some floating point numbers cannot be represented exactly.

Example: Comparison Test

This example shows how to create C code for comparison tests.

```
eJDEDBResult = JDB_InitBhvr ((void*)lpBhvrCom,
                             &hUser,
                             (JCHAR *) NULL,
                             JDEDB_COMMIT_AUTO);

/** Check for Valid hUser **/
if (eJDEDBResult == JDEDB_PASSED)
{
   statement;
}
```

Example: Creating TRUE or FALSE Test Comparison that Uses Boolean Logic

The following example is a TRUE or FALSE test comparison that uses Boolean logic:

```
/* IsStringBlank has a BOOL return type. It will always return either
 * TRUE or FALSE */
if ( IsStringBlank( szString) )
{
   statement;
}
```

Copying Strings with jdeStrcpy versus jdeStrncpy

When copying strings of the same length, such as business unit, you may use the jdeStrcpy ANSI API. If the strings differ in length-as with a description-use the jdeStrncpy ANSI API with the number of characters less one for the trailing NULL character.

```
/*********************************************************
 * Variable Definitions
 *********************************************************/
 JCHAR      szToBusinessUnit(13);
 JCHAR      szFromBusinessUnit(13);
 JCHAR      szToDescription(31);
 JCHAR      szFromDescription(41);
/*********************************************************
 * Main Processing
```

```
 ********************************************************/
   jdeStrcpy((JCHAR *) szToBusinessUnit,
             (const JCHAR *) szFromBusinessUnit );


   jdeStrncpy((JCHAR *) szToDescription,
              (const JCHAR *) szFromDescription,
              DIM(szToDescription)-1 );
```

Using the Function Clean Up Area

Use the function clean up area to release any allocated memory, including hRequest and hUser.

Example: Using the Function Clean Up Area to Release Memory

The following example shows how to release memory in the function clean up area:

```
lpdsF4301 = (LPF4301)jdeAlloc( COMMON_POOL,
                               sizeof(F4301),MEM_ZEROINIT ) ;
/************************************************************
 * Function Clean Up Section
 ************************************************************/
if (lpdsF4301 != (LPF4301 ) NULL)
{
   jdeFree( lpdsF4301 );
}

if (hRequestF4301 != (HREQUEST) NULL)
{
   JDB_CloseTable( hRequestF4301 );
}

JDB_FreeBhvr( hUser );

return ( idReturnValue ) ;
```

Inserting Function Exit Points

Where possible, use a single exit point (return) from the function. The code is more structured when a business function has a single exit point. The use of a single exit point also allows the programmer to perform cleanup, such as freeing memory and terminating ODBC requests, immediately before the return. In more complex functions, this action might be difficult or unreasonable. Include the necessary cleanup logic, such as freeing memory and terminating ODBC requests, when programming an exit point in the middle of a function.

Use the return value of the function to control statement execution. Business functions can have one of two return values: ER_SUCCESS or ER_ERROR. By initializing the return value for the function to ER_SUCCESS, the return value can be used to determine the processing flow.

Example: Inserting an Exit Point in a Function

The following example illustrates the use of a return value for the function to control statement execution:

```
  ID         idReturn        = ER_SUCCESS;
/*************************************************************
 * Main Processing
 *************************************************************/
   memset( (void *)(&dsInfo), 0x00, sizeof(DSX51013_INFO) );
   idReturn = X51013_VerifyAndRetrieveInformation( lpBhvrCom,
                                                   lpVoid,
                                                   lpDS,
                                                   &dsInfo );
 /** Check for Errors and Company or Job Level Projections **/
 if ( (idReturn == ER_SUCCESS) &&
      (lpDS->cJobCostProjections == _J('Y')) )
 {
    /** Process All Periods between the From and Thru Dates **/
    while ( (!dsInfo.bProcessed) &&
            (idReturn == ER_SUCCESS) )
    {
        /** Retrieve Calculation Information **/
        if ((dsInfo.bRetrieveBalance) && (idReturn == ER_SUCCESS))
        {
           idReturn = X51013_RetrieveAccountBalances( lpBhvrCom,
                                                      lpVoid,
                                                      lpDS,
                                                      &dsInfo );
        }
        if (idReturn == ER_SUCCESS)
        {
           statement;
        }
    } /* End Processing */
 }

/*************************************************************
 * Function Clean Up
 *************************************************************/
   if ( (dsInfo.hUser) != (HUSER) NULL )
   {
      statement;
   }

   return idReturn;
```

Terminating a Function

Always return a value with the termination of a function.

CHAPTER 6

Understanding Portability

This chapter provides overviews of portability, portability guidelines, and common server build errors and warnings.

Portability

Portability is the ability to run a program on more than one computer without modifying it. EnterpriseOne is a portable environment. This chapter presents considerations and guidelines for porting objects between systems.

Standards that affect the development of relational database systems are determined by the following:

- ANSI (American National Standards Institute) standard
- X/OPEN (European body) standard
- ISO SQL standard

Ideally, industry standards allow users to work identically with different relational database systems. Each major vendor supports industry standards but also offers extensions to enhance the functionality of the SQL language. In addition, vendors constantly release upgrades and new versions of their products.

These extensions and upgrades affect portability. Due to the effect of software development on the industry, applications need a standard interface to databases-an interface that will not be affected by differences among database vendors. When vendors provide a new release, the effect on existing applications needs to be minimal. To solve portability issues, many organizations have moved to standard database interfaces, called open database connectivity (ODBC).

Portability Guidelines

Refer to the following guidelines to develop business functions that comply with portability standards:

- Business functions must be ANSI-compatible for portability.

  Since different computer platforms might present limitations, exceptions to this rule do exist. However, do not use a non-ANSI exception without approval from the Business Function Standards Committee.

- Do not create a program that depends on data alignment because it is not portable, because each system aligns data differently by allocating bytes or words.

  For example: for a one-character field that is one byte, some systems allocate only one byte for that field; while other systems allocate the entire word for the field.

- Keep in mind that vendor libraries and function calls are system-dependent and exclusive to that vendor.

This means that if the program is compiled using a different compiler, that particular function will fail.

- Use caution when using pointer arithmetic because it is system-dependent and is based on the data alignment.
- Do not assume that all systems will initialize a variable the same way.

  Always explicitly initialize variables.
- Use caution when using the offset to retrieve the object within the data structure.

  This guideline also relates to data alignment. Use offset to define cache index.
- Always typecast if your parameter does not match the function parameter.

Note. JCHAR szArray[13] is not the same as (JCHAR *) in the function declaration. Therefore, typecast of (JCHAR *) is required for szArray for that particular function.

- Never typecast on the left-hand side of the assignment statement, as it can result in a loss of data.

  For example, in the statement `(short)nValue = (long),lValue` will lose the value of the long integer if it is too large to fit into a short integer data type.
- Do not use C++ comments (C++ comments begin with two forward slashes).

Common Server Build Errors and Warnings

PeopleSoft EnterpriseOne business functions must be ANSI-compatible for portability. Since different computer platforms and servers have their own limitations, our business functions must comply with all server standards. This topic presents guidelines for coding business functions that correctly build on different servers.

Comments within Comments

Never use comments that are included in other comments. Each "/*" should be followed by subsequent "*/". Refer to the following examples.

Example: C Comments that Comply with the ANSI Standard

Use the following C standard comment block:

```
/*****************************************************************
* Correct Method of C Comments             *
*****************************************************************/
/* SAR 1234567 Begin*/
/* Populate the lpDS->OrderedPlacedBy value from the userID only in
   the ADD mode */
   if ( lpDS->cHeaderActionCode == _J('1'))
   {
      if (IsStringBlank(lpDS->szOrderedPlacedBy))
     .{
         jdeStrcpy((JCHAR *)(lpDS->szOrderedPlacedBy),
                   (const JCHAR *)(lpDS->szUserID));
      }/* End of defaulting in the user id into Order placed by
           if the later was left blank */
   }/* SAR 1234567 End */
```

Example: Comments within Comments Cause Problems on Different Servers

The following example shows that comments within comments can cause problem on different servers:

```
/*********************************************************************
 C Comments within Comments Causing Server Build Errors and Warnings
 *********************************************************************/
/* SAR 1234567 Begin
/* Populate the lpDS->OrderedPlacedBy value from the userID only in
   the ADD mode */
*/
   if ( lpDS->cHeaderActionCode == _J('1'))
   {
      if (IsStringBlank(lpDS->szOrderedPlacedBy))
      {
         jdeStrcpy((JCHAR *)(lpDS->szOrderedPlacedBy),
                   (const JCHAR *)(lpDS->szUserID));
      }/* End of defaulting in the user id into the Order placed by
       /* if the later was left blank */
   }/* SAR 1234567 End */
```

New Line Character at the End of a Business Function

Some servers need a new line character at the end of the source and header file in order to build correctly. It is a best practice to ensure that a new line character is added at the end of each business function. Press the Enter key at the end of the code to add a new line character.

Use of Null Character

Be careful when using NULL character \0'. This character starts with a back slash. Using /0' is an error that is not reported by the compiler.

Example: Use of NULL Character

The following example shows an incorrect and a correct use of the NULL character:

```
/*************Initialize Data Structures*************************/
/*Error Code*/
/* /0' is used assuming it to be a NULL character*/
/* memset((void *)(&dsVerifyActivityRulesStatusCodeParms),
          (int)('/0'), sizeof(DSD4000260A));*/


/*Correct Use of NULL Character*/
memset((void *)(&dsVerifyActivityRulesStatusCodeParms),
       (int)('\0'), sizeof(DSD4000260A));
```

Lowercase Letters in Include Statements

When an external business function or table is included in the header file, use lowercase letters in the include statement. Uppercase letters cause build errors.

Example: Use of Lowercase Letters in Include Statements

The following example shows the incorrect and correct use of lowercase letters in the include statement:

```
/*****************************************************************
* External Business Function Header Inclusions
*****************************************************************/
/*Incorrect method of including external business function header*/
/*Include Statement Causing Build Warnings on Various Servers*/
   #include <B0000130.h>
/*Correct method of including external business function header*/
   #include <b0000130.h>
```

Initialized Variables that are Not Referenced

Each variable that is declared and initialized under the Variables Declaration section in the business function must be used in the program. For example: if the variable idReturnValue is initialized, then it must be used somewhere in the program.

CHAPTER 7

Understanding EnterpriseOne Defined Structures

EnterpriseOne provides two data types that should concern you when you create business functions: MATH_NUMERIC and JDEDATE. Since these data types might change, use the Common Library APIs provided by EnterpriseOne to manipulate them. Do not access the members of these data types directly.

This chapter provides overviews of the MATH_NUMERIC data type, and the JDEDATE data type.

MATH_NUMERIC Data Type

The MATH_NUMERIC data type is commonly used to represent numeric values in EnterpriseOne software. This data type is defined as follows:

```
struct tag MATH_NUMERIC

{
   ZCHAR String [MAXLEN_MATH_NUMERIC + 1];
   BYTE  Sign;
   ZCHAR EditCode;
   short nDecimalPosition;
   short nLength;
   WORD  wFlags;
   ZCHAR szCurrency [4];
   Short nCurrencyDecimals;
   short nPrecision;
};

typedef struct tag MATH_NUMERIC MATH_NUMERIC, FAR *LPMATH_NUMERIC;
```

The table below shows math-numeric elements and their descriptions:

MATH_NUMERIC Element	Description
String	The digits without separators
Sign	A minus sign indicates the number is negative. Otherwise, the value is 0x00.
EditCode	The data dictionary edit code used to format the number for display
nDecimalPosition	The number of digits from the right to place the decimal

MATH_NUMERIC Element	Description
nLength	The number of digits in the String
wFlags	Processing flags
szCurrency	Currency code
nCurrencyDecimals	The number of currency decimals
nPrecision	The data dictionary size

When assigning MATH_NUMERIC variables, use the MathCopy API. MathCopy copies the information, including Currency, into the location of the pointer. This API prevents any lost data in the assignment.

Initialize local MATH_NUMERIC variables with the ZeroMathNumeric API. If a MATH_NUMERIC is not initialized, invalid information, especially currency information, might be in the data structure, which can result in unexpected results at runtime.

```
/************************************************
 * Variable Definitions
 ************************************************/
   MATH_NUMERIC   mnVariable  = {0};

/************************************************
 * Main Processing
 ************************************************/
   ZeroMathNumeric( &mnVariable );
   MathCopy( &mnVariable,
             &lpDS->mnVariable );
```

JDEDATE Data Type

The JDEDATE data type is commonly used to represent dates in EnterpriseOne. The data type is defined as follows:

```
struct tag JDEDATE
{
   short nYear;
   short nMonth;
   short nDay;
};

typedef struct tag JDEDATE JDEDATE, FAR *LPJDEDATE;
```

JDEDATE Element	Description
nYear	The year

JDEDATE Element	Description
nMonth	The month
nDay	The day

Using Memcpy to Assign JDEDATE Variables

When assigning JDEDATE variables, use the memcpy function. The memcpy function copies the information into the location of the pointer. If you use a flat assignment, you might lose the scope of the local variable in the assignment, which could result in a lost data assignment.

```
/*************************************************
 * Variable Definitions
 ************************************************/
   JDEDATE   jdToDate;
/*************************************************
 * Main Processing
 ************************************************/
   memcpy((void*) &jdToDate,
          (const void *) &lpDS->jdFromDate,
          sizeof(JDEDATE) );
```

JDEDATECopy

You can use JDEDATECopy, as well as memcpy, to assign JDEDATE variables. The syntax is as follows:

```
#define JDEDATECopy(pDest, pSource)
        memcpy(pDest, pSource, sizeof(JDEDATE))
```

CHAPTER 8

Understanding Error Messages

This chapter provides overviews of error messages, inserting parameters in lpDS for error messages, standard business function error conditions, using text substitution to display specific error messages, and mapping data structure errors.

Error Messages

Messages provide an effective and usable method of communicating information to end-users. You can use simple messages or text substitution messages.

Text substitution messages provide specific information to the user. At runtime, the system replaces variables in the message with substitution values. Two types of text substitution messages exist:

- Error messages (glossary group E)
- Workflow messages (glossary group Y)

The return code from all JDB and JDE Cache APIs must be checked and an appropriate error message set, returned, or both to the calling function. The standard error messages for JDB and JDE Cache errors are shown in the following tables.

The JDB errors are:

Error ID	Description
078D	Open Table Failed
078E	Close Table Failed
078F	Insert to Table Failed
078G	Delete from Table Failed
078H	Update to Table Failed
078I	Fetch from Table Failed
078J	Select from Table Failed
078K	Set Sequence of Table Failed
078S *	Initialization of Behavior Failed

* 078S does not use text substitution

The JDE Cache errors are:

Error ID	Description
078L	Initialization of Cache Failed
078M	Open Cursor Failed
078N	Fetch from Cache Failed
078O	Add to Cache Failed
078P	Update to Cache Failed
078Q	Delete from Cache Failed
078R	Terminate of Cache Failed

Inserting Parameters in lpDS for Error Messages

Include the parameters cSuppressErrorMessage and szErrorMessageID in lpDS for error message processing. The functionality for each is described below:

- cSuppressErrorMessage (SUPPS)

  Valid data is either 1 or 0. This parameter is required if jdeErrorSet(...) is used in the business function. When cSuppressErrorMessage is set to 1, do not set an error because jdeErrorSet will automatically display an error message.

- szErrorMessageID (DTAI)

  This string contains the error message ID value that is passed back by the business function. If an error occurs in the business function, szErrorMessageID contains that error number ID.

Note. You must initialize szErrorMessageID to 4 spaces at the beginning of the function. Failure to initialize can cause memory errors.

Example: Parameters in lpDS for an Error Message

The following example includes the lpDS parameters, cSuppressErrorMessage, and szErrorMessageID:

```
if ((!IsStringBlank(lpDS->szErrorMessageID)) &&
     (lpDS->cSuppressErrorMessage != _J('1')))
 {
    jdeStrcpy ((JCHAR*) (lpDS->szErrorMessageID),
               (const JCHAR*) (_J("0653")));
    jdeErrorSet (lpBhvrCom, lpVoid, (ID) IDERRcMethodofComputation_1,
                 lpDS->szErrorMessageID, (LPVOID) NULL);
    idReturnValue = ER_ERROR;
 }

/*************************************************************
```

```
 * Function Clean Up
 ************************************************************/
   return idReturnValue;
```

Standard Business Function Error Conditions

Business functions that use the EnterpriseOne database API are required to call the Initialize Behavior function before calling any of the database functions. Set error 078S if the Initialize Behavior function does not complete successfully.

Example: Initialize Behavior Error

The following example illustrates an initialize behavior error:

```
/************************************************************************
 * Initialize Behavior
 ************************************************************************/
idJDBReturn = JDB_InitBhvr(lpBhvrCom,
                           &hUser,
                           (JCHAR *) NULL,
                           JDEDB_COMMIT_AUTO);
if (idJDBReturn != JDEDB_PASSED)
{
   jdeStrcpy (lpDS->szErrorMessageID, _J("078S"));
   if (lpDS->cSuppressErrorMessage != _J('1'))
   {
      jdeErrorSet(lpBhvrCom, lpVoid, (ID)0, _J(078S), (LPVOID) NULL);
   }
   return ER_ERROR;
}
```

Using Text Substitution to Display Specific Error Messages

You can use the EnterpriseOne text substitution APIs for returning error messages within a business function. Text substitution is a flexible method for displaying a specific error message.

Text substitution is accomplished through the data dictionary. To use text substitution, you first must set up a data dictionary item that defines text substitution for your specific error message. A selection of error messages for JDB and JDE Cache have already been created and are listed in this chapter.

Error messages for cache and tables are critical in a configurable network computing (CNC) architecture. C programmers must set the appropriate error message when working with tables or cache APIs.

JDB API errors should substitute the name of the file against which the API failed. JDE cache API errors should substitute the name of the cache for which the API failed.

When calling errors that use text substitution, you must:

- Load a data structure with the information you want to substitute in the error message.

- Call jdeErrorSet to set the error.

Example: Text Substitution in an Error Message

The following example uses text substitution in JDB_OpenTable:

```
/************************************************************
 * Open the General Ledger Table F0911
 ************************************************************/
eJDBReturn = JDB_OpenTable( hUser,
                            ID_F0911,
                            ID_F0911_DOC_TYPE__NUMBER___B,
                            idColF0911,
                            nNumColsF0911,
                            (JCHAR *)NULL,
                            &hRequestF0911);

if (eJDBReturn != JDEDB_PASSED)
{
   memset((void *)(&dsDE0022), 0x00, sizeof(dsDE0022));
   jdeStrncpy((JCHAR *)dsDE0022.szDescription,
              (const JCHAR *)(_J("F0911")),
              DIM(dsDE0022.szDescription)-1);
   jdeErrorSet (lpBhvrCom, lpVoid,(ID)0, _J("078D"), &dsDE0022);
}
```

DSDE0022 Data Structure

The data structure DSDE0022 contains the single item, szDescription[41]. Use the DSDE0022 data structure for JDB errors and JDE cache errors as the last parameter in jdeErrorSet.

Mapping Data Structure Errors with jdeCallObject

Any Business Function calling another Business Function is required to use jdeCallObject. When using jdeCallObject, be sure to match the Error IDs correctly.

The programmer needs to match the Ids from the original Business Function with the Error Ids of the Business Function in jdeCallObject. A data structure is used in the jdeCallObject to accomplish this task.

```
/************************************************************
 * Variable declarations
 ************************************************************/
CALLMAP   cm_D0000026[2]  = {{IDERRmnDisplayExchgRate_62,
                              IDERRmnExchangeRate_2}};
ID        idReturnCode    = ER_SUCCESS;  /* Return Code */
/************************************************************
* Business Function structures
*****************************************************/
DSD0000026  dsD0000026    = {0};    /* Edit Tolerance */
```

```
idReturnCode = jdeCallObject(_J("EditExchanbeRateTolerance"),
                             NULL,
                             lpBhvrCom,
                             lpVoid,
                             (void *)&dsD0000026,
                             (CALLMAP *)&cm_D0000026,
                             ND0000026,
                             (JCHAR *)NULL,
                             (JCHAR *)NULL,
                             (int)0);
```

CHAPTER 9

Understanding Data Dictionary Triggers

This chapter provides an overview of data dictionary triggers.

Data Dictionary Triggers

Data dictionary triggers are used to attach edit-and-display logic to data dictionary items. The application runtime engine executes the trigger associated with a data dictionary item at the time that the item is accessed in a form.

Custom data dictionary triggers are written in C or Named Event Rule (NER), and require a specific data structure in order to execute correctly. The custom trigger data structure is composed of three predefined members and one variable member. The predefined members are the same for every custom trigger. The variable member is different for each trigger, and it is created using the specific data element associated with the data dictionary item.

The following table shows the order of the members in the data structure along with the alias and a description of each member.

Structure Member Name	Alias	Description
idBhvrErrorId	BHVRERRID	Used by the trigger function to return the error status (ER_ERROR or ER_SUCCESS) to the application.
szBehaviorEditString	BHVREDTST	Used by the application runtime engine to pass the value for the data dictionary field to the trigger function.
szDescription001	DL01	Used by the trigger function to return the description for the value to the application.
szHomeCompany, mnAddressNumber	HMCO, AN8	Used by the trigger function to set errors (CALLMAP field).

Example: Custom Data Dictionary Trigger

An example of a custom data dictionary trigger is the edit trigger lsColumnInAddressBook. This trigger verifies that an address book record exists for the address number passed in szBehaviorEditString and, if it does, returns the alpha name in szDescription001. The variable member for this trigger is mnAddressNumber, which was created using the AN8 data dictionary item.

The C program for this trigger uses the following structure:

```
typedef struct tagDSD0100039
{
   ID              idBhvrErrorId;
   JCHAR           szBehaviorEditString[51];
   JCHAR           szDescription001[31];
   MATH_NUMERIC    mnAddressNumber;
} DSD0100039,FAR *LPDSD0100039;
```

CHAPTER 10

Understanding Unicode Compliance Standards

This chapter provides an overview of Unicode compliance standards, Unicode string functions, Unicode memory functions, pointer arithmetic, offsets, MATH_NUMERIC APIs, third-party APIs, and flat-file APIs.

Unicode Compliance Standards

The Unicode Standard is the universal character-encoding scheme for written characters and text. It defines a consistent way of way of encoding multilingual text that enables the exchange of text data internationally and creates the foundation for global software.

Facts about Unicode:

- Unicode is a very large character set containing the characters of virtually every written language.
- Unicode uses two bytes per character.

  Up to 64,000 characters can be supported using two bytes. Unicode also has a mechanism called "surrogates," which uses pairs of two bytes to describe an additional one million characters.
- 0x00 is a valid byte in a character.

  For example, the character "A" is described as 0x00 0x41, which means that normal string functions, such as strlen() and strcpy, do not work with Unicode data.

Do not use the datatype "char." Instead, use JCHAR for Unicode characters and ZCHAR for non-Unicode characters. Use ZCHAR instead of "char" in a code that needs to interface with non-Unicode APIs.

Old Syntax No Longer Available	New Syntax Non-Unicode	New Syntax Unicode
Char	ZCHAR	JCHAR
char *, PSTR	ZCHAR*, PZSTR	JCHAR*, PJSTR
'A'	_Z('A')	_J('A')
"string"	_Z("string")	_J("string")

Unicode String Functions

Two versions of all string functions exist: one for Unicode and one for non-Unicode. Naming standards for Unicode and non-Unicode string functions are as follows:

- `jdeSxxxxxx()` indicates a Unicode string function
- `jdeZSxxxx()` indicates a non-Unicode string function

Some of the replacement functions include:

Old String Functions	New String Functions Non-Unicode	New String Functions Unicode
strcpy()	jdeZStrcpy()	jdeStrcpy()
strlen()	jdeZStrlen()	jdeStrlen()
strstr()	jdeZStrstr()	jdeStrstr()
sprintf()	jdeZSprintf()	jdeSprintf()
strncpy()	jdeZStrncpy()	jdeStrncpy()

Note. The function `jdestrcpy()` was in use before the migration to Unicode. The Unicode slimer changed existing `jdestrcpy()` to `jdeStrncpyTerminate()`. Going forward, developers need to use `jdeStrncpyTerminate()` where they previously used `jdestrcpy()`.

Do not use traditional string functions, such as strcpy, strlen, and printf. All the `jdeStrxxxxxx` functions explicitly handle strings, so use character length instead of the `sizeof()` operator, which returns a byte count.

When using `jdeStrncpy()`, the third parameter is the number of characters, not the number of bytes.

The `DIM()` macro gives the number of characters of an array. Given "`JCHAR a[10];`", `DIM(a)` returns 10, while `sizeof(a)` returns 20. "`strncpy (a, b, sizeof (a));`" needs to become "`jdeStrncpy (a, b, DIM (a));`".

Example: Using Unicode String Functions

The following example shows how to use Unicode string functions:

```
/*********************************************************************
 In this example jdeStrncpy replaces strncpy. Also sizeof is
 replaced by DIM.
 *********************************************************************/
/* Set key to F38112 */

/*Unicode Compliant*/
jdeStrncpy(dsKey1F38112.dxdcto,
          (const JCHAR *)(dsF4311ZDetail->pwdcto),
          DIM(dsKey1F38112.dxdcto) - 1);
```

Unicode Memory Functions

The memset() function changes memory byte by byte. For example, `memset (buf, ' ', sizeof (buf));` sets the 10 bytes pointed to by the first argument, buf, to the value 0x20, the space character. Since a Unicode character is 2 bytes, each character is set to 0x2020, which is the dagger character (†) in Unicode.

A new function, jdeMemset() sets memory character by character rather than byte by byte. This function takes a void pointer, a JCHAR, and the number of bytes to set. Use `jdeMemset (buf, _J (' '), sizeof (buf));` to set the Unicode string buf so that each character is 0x0020. When using jdeMemset(), the third parameter, sizeof(buf), is the number of bytes, not characters.

Note. You can use memset when filling a memory block with NULL. For all other characters, use jdeMemset. You also can use jdeMemset for a NULL character.

Example: Using jdeMemset when Setting Characters to Values other than NULL

The following example shows how to use jdeMemset when setting characters to values other than NULL:

```
/*********************************************************************
 In this example memset is replaced by jdeMemset. We need to change
 memset to jdeMemset because we are setting each character of the
 string to a value other than NULL. Also, because jdeMemset works in
 bytes, we cannot just subtract 1 from sizeof(szSubsidiaryBlank) to
 prevent the last character from being set to  .  We must multiply
 1 by sizeof(JCHAR).
 *********************************************************************/

/*Unicode Compliant*/
jdeMemset((void *)(szSubsidiaryBlank), _J(' '),
          (sizeof(szSubsidiaryBlank) - (1*sizeof(JCHAR))));
```

Pointer Arithmetic

When advancing a JCHAR pointer, it is important to advance the pointer by the correct number. In the example below, the intent is to initialize each member of an array consisting of JCHAR strings to blank. Inside the "For" loop, the pointer is advanced to point to the next member of the array of JCHAR strings after assigning a value to one of the members of the array. This is achieved by adding the maximum length of the string to the pointer. Since pStringPtr has been defined as a pointer to a JCHAR, adding MAXSTRLENGTH to pStringPtr results in pStringPtr pointing to the next member of the array of strings.

```
#define MAXSTRLENGTH 10
JCHAR             *pStringPtr;
LPMATH_NUMERIC    pmnPointerToF3007;
for(i=(iDayOfTheWeek+iNumberOfDaysInMonth);i<CALENDARDAYS;i++)
{
  FormatMathNumeric(pStringPtr, &pmnPointerToF3007[i]);
  pStringPtr = pStringPtr + MAXSTRLENGTH;
}
```

The following tables illustrate the effect of adding MAXSTRLENGTH to pStringPtr. The top row in both tables contains memory locations; the bottom rows contain the contents of those memory locations.

The arrow indicates the memory location that pStringPtr points to before MAXSTRLENGTH is added to pStringPtr.

∇																			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
00	49	00	52	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20

21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
00	49	00	52	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20

The arrow below indicates the memory location that`pStringPtr`points to after`MAXSTRLENGTH`is added to`pStringPtr`. Adding 10 to`pStringPtr`makes it move 20 bytes, as it has been declared of type`JCHAR`.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
00	52	00	53	00	54	00	20	00	20	00	20	00	20	00	20	00	20	00	20

∇																			
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
00	49	00	52	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20

If `pStringPtr` is advanced by the value`MAXSTRLENGTH * sizeof(JCHAR)`, then `pStringPtr` advances twice as much as intended and results in memory corruption.

Offsets

When adding an offset to a pointer to derive the location of another variable or entity, it is important to determine the method in which the offset was initially created.

In the following example, `lpKeyStruct->CacheKey[n].nOffset` is added to `lpData` to arrive at the location of a Cache Key segment. This offset was for the segment created using the ANSI C function `offsetof`, which returns the number of bytes. Therefore, to arrive at the location of Cache Key segment, cast the data structure pointer to type BYTE.

```
lpTemp1 = (BYTE *)lpData + lpKeyStruct->CacheKey[n].nOffset;
lpTemp2 = (BYTE *)lpKey  + lpKeyStruct->CacheKey[n].nOffset;
```

In a non-Unicode environment, `lpData` could have been cast to be of type `CHAR *` as character size is one Byte in a non-Unicode environment. In a Unicode environment, however, `lpData` has to be explicitly cast to be of type `(JCHAR *)` since size of a `JCHAR` is 2 bytes.

MATH_NUMERIC APIs

The string members of the MATH_NUMERIC data structure are in ZCHAR (non-Unicode) format. The EnterpriseOne Common Library API includes several functions that retrieve and manipulate these strings in both JCHAR (Unicode) and ZCHAR (non-Unicode) formats.

To retrieve the string value of a MATH_NUMERIC data type in JCHAR format, use the FormatMathNumeric API function. The following example illustrates the use of this function:

```
/* Declare variables */
JCHAR      szJobNumber[MAXLEN_MATH_NUMERIC+1] = _J("\0");
/* Retrieve the string value of the job number */
FormatMathNumeric(szJobNumber, &lpDS->mnJobnumber);
```

To retrieve the string value of a MATH_NUMERIC data type in ZCHAR format, use the jdeMathGetRawString API function. The following example illustrates the use of this function:

```
/* Declare variables */
ZCHAR      zzJobNumber[MAXLEN_MATH_NUMERIC+1] = _Z("\0");
/* Retrieve the string value of the job number */
zzJobNumber = jdeMathGetRawString(&lpDS->mnJobnumber);
```

Another commonly used MATH_NUMERIC API function is jdeMathSetCurrencyCode. This function is used to update the currency code member of a MATH_NUMERIC data structure. Two versions of this function exist: jdeMathCurrencyCode and jdeMathCurrencyCodeUNI. The jdeMathCurrencyCode function is used to update the currency code with a ZCHAR value, and jdeMathCurrencyCodeUNI is used to update the currency code with a JCHAR value. The following example illustrates the use of these two functions:

```
/* Declare variables */
ZCHAR      zzCurrencyCode[4] = _Z("USD");
JCHAR      szCurrencyCode[4] = _J("USD");
/* Set the currency code using a ZCHAR value */
jdeMathSetCurrencyCode(&lpDs->mnAmount, (ZCHAR *) zzCurrencyCode);
/* Set the currency code using a JCHAR value /*
jdeMathSetCurrencyCodeUNI(&lpDS->mnAmount, (JCHAR *) szCurrencyCode);
```

Third-Party APIs

Some third-party program interfaces (APIs) do not support Unicode character strings. In these cases, you must convert character strings to non-Unicode format before calling the API, and convert them back to Unicode format for storage in EnterpriseOne. Use the following guidelines when programming for a non-Unicode API:

- Declare a Unicode and a non-Unicode variable for each API string parameter.
- Convert the Unicode strings to non-Unicode strings before calling the API.
- Call the API passing the non-Unicode strings in the parameter list.
- Convert the returned non-Unicode strings to Unicode strings for storage in EnterpriseOne.

Example: Third-Party API

The following example calls a third-party API named GetStateName that accepts a two-character state code and returns a 30-character state name:

```
/* Declare variables */
JCHAR  szStateCode[3] = _J("CO");  /* Unicode state code */
JCHAR  szStateName[31] = _J("\0");  /* Unicode state name */
ZCHAR  zzStateCode[3] = _Z("\0");  /* Non-Unicode state code */
ZCHAR  zzStateName[31] = _Z("\0");  /* Non-Unicode state name */
BOOL  bReturnStatus = FALSE;    /* API return flag */
/* Convert unicode strings to non-unicode strings */
jdeFromUnicode(zzStateCode, szStateCode, DIM(zzStateCode), NULL);
/* Call API */
bReturnStatus = GetStateName(zzStateCode, zzStateName);
/* Convert non-unicode strings to unicode strings for storage in
* EnterpriseOne */
jdeToUnicode(szStateName, zzStateName, DIM(szStateName), NULL);
```

Flat-File APIs

EnterpriseOne APIs such as jdeFprintf() convert data. This means that the default flat file I/O for character data is in Unicode. If the users of EnterpriseOne-generated flat files are not Unicode enabled, they will not be able to read the flat file correctly. Therefore, use an additional set of APIs.

An interactive application allows users to configure flat file encoding based on attributes such as application name, application version name, user name, and environment name. The API set includes the following file I/O functionalities: fwrite/fread, fprintf/fscanf, fputs/fgets, and fputc/fgetc. The API converts the data using the code page specified in the configuration application. One additional parameter, lpBhvrCom, must be passed to the functions so that the conversion function can find the configuration for that application or version.

These new APIs only need to be called if a process outside of EnterpriseOne is writing or reading the flat file data. If the file is simply a work file or a debugging file and will be written and read by EnterpriseOne, use the non-converting APIs (for example, jdeFprintf()).

Example: Flat-File APIs

The following example writes text to a flat file that would only be read by EnterpriseOne. Encoding in the file will be Unicode.

```
FILE *fp;
fp = jdeFopen(_J( c:/testBSFNZ.txt), _J(w+));
jdeFprintf(fp, _J("%s%d\n"), _J("Line "), 1);
jdeFclose(fp);
```

The following example writes text to a flat file that would be read by third-party systems. Encoding in the file will be based on the encoding configured.

```
FILE *fp;
fp = jdeFopen(_J( c:/testBSFNZ.txt), _J(w+));
jdeFprintfConvert(lpBhvrCom, fp, _J("%s%d\n"), _J("Line "), 1);
jdeFclose(fp);
```

CHAPTER 11

Understanding Standard Header and Source Files

Business Function Design generates the .c and .h templates. The following sections detail what information goes into each section of the templates and refer you to related information.

Model source code files exist for both the .c and .h modules of your business function.

This chapter provides an overview of the standard header and the standard source.

Standard Header

Header files help the compiler properly create a business function. The C language contains 33 keywords. Everything else, such as printf and getchar, is a function. Functions are defined in header files that you include at the beginning of a business function. Without header files, the compiler does not recognize the functions and might return error messages.

The following example shows the standard header for a business function source file:

```
/*****************************************************************
 *  Header File: BXXXXXXX.h
 *  Description: Generic Business Function Header File
 *    History:
 *     Date    Programmer SAR# - Description
 *     ---------- ---------- ---------------------------
 *  Author 06/06/2005           - Created
 *
 * Copyright (c) PeopleSoft, Inc., 2004
 *
 * This unpublished material is proprietary to PeopleSoft, Inc.
 * All rights reserved. The methods and
 * techniques described herein are considered trade secrets
 * and/or confidential. Reproduction or distribution, in whole
 * or in part, is forbidden except by express written permission
 * of PeopleSoft, Inc.
 *****************************************************************/
#ifndef __BXXXXXXX_H
#define __BXXXXXXX_H
/*****************************************************************
 * Table Header Inclusions
 *****************************************************************/

/*****************************************************************
```

```
 * External Business Function Header Inclusions
 ************************************************************/

/************************************************************
 * Global Definitions
 ************************************************************/

/************************************************************
 * Structure Definitions
 ************************************************************/

/************************************************************
 * DS Template Type Definitions
 ************************************************************/

/************************************************************
 * Source Preprocessor Definitions
 ************************************************************/
#if defined (JDEBFRTN)
   #undef JDEBFRTN
#endif

#if defined (WIN32)
   #if defined (WIN32)
      #define JDEBFRTN(r) __declspec(dllexport) r
   #else
      #define JDEBFRTN(r) __declspec(dllimport) r
   #endif
#else
   #define JDEBFRTN(r) r
#endif
/************************************************************
 * Business Function Prototypes
 ************************************************************/
JDEBFRTN (ID) JDEBFWINAPI GenericBusinessFunction
                (LPBHVRCOM     lpBhvrCom,
                 LPVOID        lpVoid,
                 LPDSDXXXXXXXX lpDS);

/************************************************************
 * Internal Function Prototypes
 ************************************************************/
#endif  /* ___BXXXXXXX_H */
```

Business Function Name and Description

Use the Business Function Name and Description section to define the name of the business function, describe the business function, and maintain the modification log.

Copyright Notice

The Copyright section contains the PeopleSoft, Inc. copyright notice and must be included in each source file. Do not change this section.

Header Definition for a Business Function

The Header Definition for a Business Function section contains the "#define" of the business function. It is generated by the tool. Do not change this section.

Table Header Inclusions

The Table Header Inclusions section includes the table headers associated with tables directly accessed by the business function.

See Also

Chapter 6, “Understanding Portability,” Lowercase Letters in Include Statements, page 35

External Business Function Header Inclusions

The External Business Function Header Inclusions section contains the business function headers associated with externally defined business functions that are directly accessed by the business function.

See Also

Chapter 6, “Understanding Portability,” Lowercase Letters in Include Statements, page 35

Global Definitions

Use the Global Definitions section to define global constants used by the business function. Enter names in uppercase, separated by an underscore.

See Also

Chapter 4, “Understanding Declaring and Initializing Variables and Data Structures,” Using Define Statements, page 13

Structure Definitions

Define structures used by the business function in the Structure Definitions section. Structure names should be prefixed by the Source File Name to prevent conflicts with structures of the same name in other business functions.

See Also

Chapter 2, “Understanding Naming Conventions,” page 3

Chapter 4, “Understanding Declaring and Initializing Variables and Data Structures,” Using Typedef Statements, page 14

DS Template Type Definitions

Do not modify the DS Template Type Definitions section. The DS Template Type Definitions section defines the business functions contained in the source that correspond to the header. The structure is generated from the business function or data structure design window in Object Management Workbench.

Source Preprocessing Definitions

The Source Preprocessing Definitions section defines the entry point of the business function and includes the opening bracket required by C functions. Do not change this section.

Business Function Prototypes

Use the Business Function Prototypes section to prototype the functions defined in the source file.

See Also

Chapter 4, "Understanding Declaring and Initializing Variables and Data Structures," Creating Function Prototypes, page 15

Internal Function Prototypes

The Internal Function Prototypes section contains a description and parameters of the function.

See Also

Chapter 2, "Understanding Naming Conventions," page 3

Chapter 4, "Understanding Declaring and Initializing Variables and Data Structures," Creating Function Prototypes, page 15

Standard Source

The source file contains instructions for the business function. The following sections describe the sections of the standard source.

A template generated for a standard source file when you create a EnterpriseOne business function appears in the following pages:

```
#include <jde.h>
#define bxxxxxxx_c
/**************************************************************
 *  Source File: bxxxxxxx
 *
 *  Description: Generic Business Function Source File
 *
 *    History:
 *     Date    Programmer SAR# - Description
 *    ----------------------------------------------------
 *  Author 06/06/2005           - Created
 *
```

```
 * Copyright (c) PeopleSoft, Inc., 2005
 *
 * This unpublished material is proprietary to PeopleSoft, Inc.
 * All rights reserved. The methods and techniques described
 * herein are considered trade secrets and/or confidential.
 * Reproduction or distribution, in whole or in part, is
 * forbidden except by express written permission of
 * PeopleSoft, Inc.
 ****************************************************************/
/****************************************************************
 * Notes:
 *
 ****************************************************************/

#include <bxxxxxxx.h>
/****************************************************************
 * Global Definitions

****************************************************************/

/****************************************************************
 * Business Function: GenericBusinessFunction
 *
 *   Description: Generic Business Function
 *
 *   Parameters:
 *   LPBHVRCOM  lpBhvrCom Business Function Communications
 *   LPVOID    lpVoid  Void Parameter - DO NOT USE!
 *   LPDSDXXXXXX lpDS   Parameter Data Structure Pointer
 *
 ****************************************************************/
JDEBFRTN (ID) JDEBFWINAPI GenericBusinessFunction
                 (LPBHVRCOM     lpBhvrCom,
                  LPVOID        lpVoid,
                  LPDSDXXXXXXX lpDS)
{
 /*************************************************************
  * Variable declarations
  *************************************************************/

 /*************************************************************
  * Declare structures
  *************************************************************/

 /*************************************************************
  * Declare pointers
  *************************************************************/

 /*************************************************************
  * Check for NULL pointers
```

```
 ************************************************************/
if ((lpBhvrCom == (LPBHVRCOM) NULL) ||
    (lPVoid  == (LPVOID)  NULL) ||
    (lpDS   == (LPDSDXXXXXXX) NULL))
{
   jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0,
                4363, (LPVOID) NULL);
   return ER_ERROR;
}
/************************************************************
 * Set pointers
 ************************************************************/

/************************************************************
 * Main Processing
 ************************************************************/

/************************************************************
 * Function Clean Up
 ************************************************************/

return (ER_SUCCESS);
}
/* Internal function comment block */
/************************************************************
 *  Function: Ixxxxxxx_a // Replace xxxxxxx with source file
 *             // number
 *             // and a with the function name
 *   Notes:
 *
 *  Returns:
 *
 * Parameters:
************************************************************/
```

Business Function Name and Description

Use this section to maintain the name and description of the business function. Also use this section to maintain the modification log.

Copyright Notice

The Copyright section contains the PeopleSoft, Inc. copyright notice and must be included in each source file. Do not make any changes to this section.

Notes

Use the Notes section to include information for anyone who might review the code in the future. For example, describe any peculiarities associated with the business function or any special logic.

Global Definitions

Use the Global Definitions section to define global constants used by the business function.

See Also

Chapter 4, “Understanding Declaring and Initializing Variables and Data Structures,” Initializing Variables, page 16

Header File for Associated Business Function

In the Header File for Associated Business Function section, include the header file associated with the business function using #include. If you need to include additional header files in the source, place them here.

Business Function Header

The Business Function Header section contains a description of each of the parameters used by the business function. Do not make any changes to this section.

Variable Declarations

The Variable Declarations section defines all required function variables. For ease of use, define the variables sequentially by type.

See Also

Chapter 2, “Understanding Naming Conventions,” page 3

Chapter 4, “Understanding Declaring and Initializing Variables and Data Structures,” Initializing Variables, page 16

Declare Structures

Define any structures that are required by the function in the Declare Structures section.

See Also

Chapter 4, “Understanding Declaring and Initializing Variables and Data Structures,” Creating Function Prototypes, page 15

Pointers

If any pointers are required by the function, define them in the Pointers section. Name the pointer so that it reflects the structure to which it is pointing. For example, lpDS1100 is pointing to the structure DS1100.

Check for NULL Pointers

The Check for NULL Pointers section checks for parameter pointers that are NULL. Do not change this section.

Set Pointers

Use the Set Pointers section if you did not initialize the variables when declaring them. You must assign values to all pointers that you define.

See Also

Chapter 4, “Understanding Declaring and Initializing Variables and Data Structures,” Creating Function Prototypes, page 15

Main Processing

Use the Main Processing section to write your code.

Function Clean Up

Use the Function Clean Up section to release any allocated memory.

See Also

Chapter 5, “Understanding General Coding Guidelines,” Using the Function Clean Up Area, page 30

Internal Function Comment Block

The Internal Function Comment Block section contains a description and parameters of the function.

See Also

Chapter 2, “Understanding Naming Conventions,” page 3

Chapter 4, “Understanding Declaring and Initializing Variables and Data Structures,” Creating Function Prototypes, page 15

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.

academic career In PeopleSoft Enterprise Campus Solutions, all course work that a student undertakes at an academic institution and that is grouped in a single student record. For example, a university that has an undergraduate school, a graduate school, and various professional schools might define several academic careers—an undergraduate career, a graduate career, and separate careers for each professional school (law school, medical school, dental school, and so on).

academic institution In PeopleSoft Enterprise Campus Solutions, an entity (such as a university or college) that is independent of other similar entities and that has its own set of rules and business processes.

academic organization In PeopleSoft Enterprise Campus Solutions, an entity that is part of the administrative structure within an academic institution. At the lowest level, an academic organization might be an academic department. At the highest level, an academic organization can represent a division.

academic plan In PeopleSoft Enterprise Campus Solutions, an area of study—such as a major, minor, or specialization—that exists within an academic program or academic career.

academic program In PeopleSoft Enterprise Campus Solutions, the entity to which a student applies and is admitted and from which the student graduates.

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.

address usage In PeopleSoft Enterprise Campus Solutions, a grouping of address types defining the order in which the address types are used. For example, you might define an address usage code to process addresses in the following order: billing address, dormitory address, home address, and then work address.

adjustment calendar In PeopleSoft Enterprise Campus Solutions, the adjustment calendar controls how a particular charge is adjusted on a student's account when the student drops classes or withdraws from a term. The charge adjustment is based on how much time has elapsed from a predetermined date, and it is determined as a percentage of the original charge amount.

administrative function In PeopleSoft Enterprise Campus Solutions, a particular functional area that processes checklists, communication, and comments. The administrative function identifies which variable data is added to a person's checklist or communication record when a specific checklist code, communication category, or comment is assigned to the student. This key data enables you to trace that checklist, communication, or comment back to a specific processing event in a functional area.

admit type In PeopleSoft Enterprise Campus Solutions, a designation used to distinguish first-year applications from transfer applications.

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.

analysis database In PeopleSoft Enterprise Campus Solutions, database tables that store large amounts of student information that may not appear in standard report formats. The analysis database tables contain keys for all objects in a report that an application program can use to reference other student-record objects that are not contained in the printed report. For instance, the analysis database contains data on courses that are considered for satisfying a requirement but that are rejected. It also contains information on

	courses captured by global limits. An analysis database is used in PeopleSoft Enterprise Academic Advisement.
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.
audience	In PeopleSoft Enterprise Campus Solutions, a segment of the database that relates to an initiative, or a membership organization that is based on constituent attributes rather than a dues-paying structure. Examples of audiences include the Class of '65 and Undergraduate Arts & Sciences.
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.
billing career	In PeopleSoft Enterprise Campus Solutions, the one career under which other careers are grouped for billing purposes if a student is active simultaneously in multiple careers.
bio bit or bio brief	In PeopleSoft Enterprise Campus Solutions, a report that summarizes information stored in the system about a particular constituent. You can generate standard or specialized reports.
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.

campus In PeopleSoft Enterprise Campus Solutions, an entity that is usually associated with a distinct physical administrative unit, that belongs to a single academic institution, that uses a unique course catalog, and that produces a common transcript for students within the same academic career.

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.

category In PeopleSoft Enterprise Campus Solutions, a broad grouping to which specific comments or communications (contexts) are assigned. Category codes are also linked to 3C access groups so that you can assign data-entry or view-only privileges across functions.

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.

checklist code In PeopleSoft Enterprise Campus Solutions, a code that represents a list of planned or completed action items that can be assigned to a staff member, volunteer, or unit. Checklists enable you to view all action assignments on one page.

class In PeopleSoft Enterprise Campus Solutions, a specific offering of a course component within an academic term.

See also *course.*

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.*

clearance In PeopleSoft Enterprise Campus Solutions, the period of time during which a constituent in PeopleSoft Contributor Relations is approved for involvement in an initiative or an action. Clearances are used to prevent development officers from making multiple requests to a constituent during the same time period.

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.

cohort In PeopleSoft Enterprise Campus Solutions, the highest level of the three-level classification structure that you define for enrollment management. You can define a cohort level, link it to other levels, and set enrollment target numbers for it.

See also *population*and *division*.

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.

comm key See *communication key.*

communication key In PeopleSoft Enterprise Campus Solutions, a single code for entering a combination of communication category, communication context, communication method, communication direction, and standard letter code. Communication keys (also called *comm keys* or *speed keys*) can be created for background processes as well as for specific users.

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.

constituents In PeopleSoft Enterprise Campus Solutions, friends, alumni, organizations, foundations, or other entities affiliated with the institution, and about which the institution maintains information. The constituent types delivered with PeopleSoft Enterprise Contributor Relations Solutions are based on those defined by the Council for the Advancement and Support of Education (CASE).

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 Campus Solutions, a specific instance of a comment or communication. One or more contexts are assigned to a category, which you link to 3C access groups so that you can assign data-entry or view-only privileges across functions.

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.

course In PeopleSoft Enterprise Campus Solutions, a course that is offered by a school and that is typically described in a course catalog. A course has a standard syllabus and credit level; however, these may be modified at the class level. Courses can contain multiple components such as lecture, discussion, and lab.

See also *class*.

course share set In PeopleSoft Enterprise Campus Solutions, a tag that defines a set of requirement groups that can share courses. Course share sets are used in PeopleSoft Enterprise Academic Advisement.

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.
division	In PeopleSoft Enterprise Campus Solutions, the lowest level of the three-level classification structure that you define in PeopleSoft Enterprise Recruiting and Admissions for enrollment management. You can define a division level, link it to other levels, and set enrollment target numbers for it. See also *population* and *cohort*.
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.
equity item limit	In PeopleSoft Enterprise Campus Solutions, the amounts of funds set by the institution to be awarded with discretionary or gift funds. The limit could be reduced by amounts equal to such things as expected family contribution (EFC) or parent contribution. Students are packaged by Equity Item Type Groups and Related Equity Item Types. This limit can be used to assure that similar student populations are packaged equally.

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.

financial aid term In PeopleSoft Enterprise Campus Solutions, a combination of a period of time that the school determines as an instructional accounting period and an academic career. It is created and defined during the setup process. Only terms eligible for financial aid are set up for each financial aid career.

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.

gap In PeopleSoft Enterprise Campus Solutions, an artificial figure that sets aside an amount of unmet financial aid need that is not funded with Title IV funds. A gap can be used to prevent fully funding any student to conserve funds, or it can be used to preserve unmet financial aid need so that institutional funds can be awarded.

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.

gift table In PeopleSoft Enterprise Campus Solutions, a table or so-called *donor pyramid* describing the number and size of gifts that you expect will be needed to successfully complete the campaign in PeopleSoft Contributor Relations. The gift table enables you to estimate the number of donors and prospects that you need at each gift level to reach the campaign goal.

GL business unit Abbreviation for *general ledger business unit.* A unit in an organization that is an independent entity for accounting purposes. It maintains its own set of accounting books.

See also *business unit*.

GL entry template Abbreviation for *general ledger entry template.* In PeopleSoft Enterprise Campus Solutions, a template that defines how a particular item is sent to the general ledger. An item-type maps to the general ledger, and the GL entry template can involve multiple general ledger accounts. The entry to the general ledger is further controlled

by high-level flags that control the summarization and the type of accounting—that is, accrual or cash.

GL Interface process Abbreviation for *General Ledger Interface process.* In PeopleSoft Enterprise Campus Solutions, a process that is used to send transactions from PeopleSoft Enterprise Student Financials to the general ledger. Item types are mapped to specific general ledger accounts, enabling transactions to move to the general ledger when the GL Interface process is run.

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.

initiative In PeopleSoft Enterprise Campus Solutions, the basis from which all advancement plans are executed. It is an organized effort targeting a specific constituency, and it can occur over a specified period of time with specific purposes and goals. An initiative can be a campaign, an event, an organized volunteer effort, a membership drive, or any other type of effort defined by the institution. Initiatives can be multipart, and they can be related to other initiatives. This enables you to track individual parts of an initiative, as well as entire initiatives.

inquiry access In PeopleSoft Enterprise Campus Solutions, a type of security access that permits the user only to view data.

See also *update access.*

institution In PeopleSoft Enterprise Campus Solutions, an entity (such as a university or college) that is independent of other similar entities and that has its own set of rules and business processes.

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.

item shuffle In PeopleSoft Enterprise Campus Solutions, a process that enables you to change a payment allocation without having to reverse the payment.

joint communication In PeopleSoft Enterprise Campus Solutions, one letter that is addressed jointly to two people. For example, a letter might be addressed to both Mr. Sudhir Awat and Ms. Samantha Mortelli. A relationship must be established between the two individuals in the database, and at least one of the individuals must have an ID in the database.

keyword In PeopleSoft Enterprise Campus Solutions, a term that you link to particular elements within PeopleSoft Student Financials, Financial Aid, and Contributor Relations. You can use keywords as search criteria that enable you to locate specific records in a search dialog box.

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.

LMS Abbreviation for *learning management system.* In PeopleSoft Enterprise Campus Solutions, LMS is a PeopleSoft Student Records feature that provides a common set of interoperability standards that enable the sharing of instructional content and data between learning and administrative environments.

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.

mass change In PeopleSoft Enterprise Campus Solutions, mass change is a SQL generator that can be used to create specialized functionality. Using mass change, you can set up a series of Insert, Update, or Delete SQL statements to perform business functions that are specific to the institution.

See also *3C engine*.

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.

need In PeopleSoft Enterprise Campus Solutions, the difference between the cost of attendance (COA) and the expected family contribution (EFC). It is the gap between the cost of attending the school and the student's resources. The financial aid package is based on the amount of financial need. The process of determining a student's need is called *need analysis.*

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.

payment shuffle In PeopleSoft Enterprise Campus Solutions, a process allowing payments that have been previously posted to a student's account to be automatically reapplied when a higher priority payment is posted or the payment allocation definition is changed.

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.

person of interest A person about whom the organization maintains information but who is not part of the workforce.

personal portfolio In PeopleSoft Enterprise Campus Solutions, the user-accessible menu item that contains an individual's name, address, telephone number, and other personal information.

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.

population In PeopleSoft Enterprise Campus Solutions, the middle level of the three-level classification structure that you define in PeopleSoft Enterprise Recruiting and Admissions for enrollment management. You can define a population level, link it to other levels, and set enrollment target numbers for it.

See also *division* and *cohort*.

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.

primacy number In PeopleSoft Enterprise Campus Solutions, a number that the system uses to prioritize financial aid applications when students are enrolled in multiple academic careers and academic programs at the same time. The Consolidate Academic Statistics process uses the primacy number indicated for both the career and program at the institutional level to determine a student's primary career and program. The system also uses the

	number to determine the primary student attribute value that is used when you extract data to report on cohorts. The lowest number takes precedence.
primary name type	In PeopleSoft Enterprise Campus Solutions, the name type that is used to link the name stored at the highest level within the system to the lower-level set of names that an individual provides.
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.
prospects	In PeopleSoft Enterprise Campus Solutions, students who are interested in applying to the institution. In PeopleSoft Enterprise Contributor Relations, individuals and organizations that are most likely to make substantial financial commitments or other types of commitments to the institution.
publishing	In PeopleSoft Enterprise Incentive Management, a stage in processing that makes incentive-related results available to participants.

rating components In PeopleSoft Enterprise Campus Solutions, variables used with the Equation Editor to retrieve specified populations.

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.*

recname The name of a record that is used to determine the associated field to match a value or set of values.

recognition In PeopleSoft Enterprise Campus Solutions, the recognition type indicates whether the PeopleSoft Enterprise Contributor Relations donor is the primary donor of a commitment or shares the credit for a donation. Primary donors receive hard credit that must total 100 percent. Donors that share the credit are given soft credit. Institutions can also define other share recognition-type values such as memo credit or vehicle credit.

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.

reversal indicator In PeopleSoft Enterprise Campus Solutions, an indicator that denotes when a particular payment has been reversed, usually because of insufficient funds.

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.

search/match In PeopleSoft Enterprise Campus Solutions and PeopleSoft Enterprise Human Resources Management Solutions, a feature that enables you to search for and identify duplicate records in the database.

seasonal address In PeopleSoft Enterprise Campus Solutions, an address that recurs for the same length of time at the same time of year each year until adjusted or deleted.

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.

service impact In PeopleSoft Enterprise Campus Solutions, the resulting action triggered by a service indicator. For example, a service indicator that reflects nonpayment of account balances by a student might result in a service impact that prohibits registration for classes.

service indicator In PeopleSoft Enterprise Campus Solutions, indicates services that may be either withheld or provided to an individual. Negative service indicators indicate holds that prevent the individual from receiving specified services, such as check-cashing privileges or registration for classes. Positive service indicators designate special services that are provided to the individual, such as front-of-line service or special services for disabled students.

session In PeopleSoft Enterprise Campus Solutions, time elements that subdivide a term into multiple time periods during which classes are offered. In PeopleSoft Contributor Relations, a session is the means of validating gift, pledge, membership, or adjustment data entry . It controls access to the data entered by a specific user ID. Sessions are balanced, queued, and then posted to the institution's financial system. Sessions must be posted to enter a matching gift or pledge payment, to make an adjustment, or to process giving clubs or acknowledgements.

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 key process In PeopleSoft Enterprise Campus Solutions, a process that relates a particular transaction to the source of the charge or financial aid. On selected pages, you can drill down into particular charges.

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.

speed key See *communication key.*

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.

standard letter code In PeopleSoft Enterprise Campus Solutions, a standard letter code used to identify each letter template available for use in mail merge functions. Every letter generated in the system must have a standard letter code identification.

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.

tax authority In PeopleSoft Enterprise Campus Solutions, a user-defined element that combines a description and percentage of a tax with an account type, an item type, and a service impact.

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.

3C engine Abbreviation for *Communications, Checklists, and Comments engine.* In PeopleSoft Enterprise Campus Solutions, the 3C engine enables you to automate business processes that involve additions, deletions, and updates to communications, checklists,

	and comments. You define events and triggers to engage the engine, which runs the mass change and processes the 3C records (for individuals or organizations) immediately and automatically from within business processes.
3C group	Abbreviation for *Communications, Checklists, and Comments group.* In PeopleSoft Enterprise Campus Solutions, a method of assigning or restricting access privileges. A 3C group enables you to group specific communication categories, checklist codes, and comment categories. You can then assign the group inquiry-only access or update access, as appropriate.
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.
tuition lock	In PeopleSoft Enterprise Campus Solutions, a feature in the Tuition Calculation process that enables you to specify a point in a term after which students are charged a minimum (or *locked*) fee amount. Students are charged the locked fee amount even if they later drop classes and take less than the normal load level for that tuition charge.
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.
update access	In PeopleSoft Enterprise Campus Solutions, a type of security access that permits the user to edit and update data. See also *inquiry access*.
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.

worker A person who is part of the workforce; an employee or a contingent worker.

workset A group of people and organizations that are linked together as a set. You can use worksets to simultaneously retrieve the data for a group of people and organizations and work with the information on a single page.

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

W