



EnterpriseOne 8.93 Development Standards Business Function Programming PeopleBook

May 2004

EnterpriseOne 8.93
Development Standards Business Function Programming PeopleBook
SKU TT893DSB0504

Copyright© 2004 PeopleSoft, Inc. All rights reserved.

All material contained in this documentation is proprietary and confidential to PeopleSoft, Inc. ("PeopleSoft"), protected by copyright laws and subject to the nondisclosure provisions of the applicable PeopleSoft agreement. No part of this documentation may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, including, but not limited to, electronic, graphic, mechanical, photocopying, recording, or otherwise without the prior written permission of PeopleSoft.

This documentation is subject to change without notice, and PeopleSoft does not warrant that the material contained in this documentation is free of errors. Any errors found in this document should be reported to PeopleSoft in writing.

The copyrighted software that accompanies this document is licensed for use only in strict accordance with the applicable license agreement which should be read carefully as it governs the terms of use of the software and this document, including the disclosure thereof.

PeopleSoft, PeopleTools, PS/nVision, PeopleCode, PeopleBooks, PeopleTalk, and Vantive are registered trademarks, and Pure Internet Architecture, Intelligent Context Manager, and The Real-Time Enterprise are trademarks of PeopleSoft, Inc. All other company and product names may be trademarks of their respective owners. The information contained herein is subject to change without notice.

Open Source Disclosure

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.

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.

Table of Contents

About These EnterpriseOne PeopleBooks Preface	1
EnterpriseOne Application Prerequisites	1
Obtaining Documentation Updates	1
Typographical Conventions and Visual Cues	2
Typographical Conventions.....	2
Visual Cues	3
Comments and Suggestions	3
Business Function Programming Standards	4
Program Flow	5
Naming Conventions	6
Source and Header File Names	6
Function Names	7
Variable Names.....	7
Example: Hungarian Notation for Variable Names	8
Business Function Data Structure Names	9
Readability	10
Maintaining the Source and Header Code Change Log	10
Inserting Comments	10
Example: Inserting Comments	11
Indenting Code	11
Example: Indenting Code.....	12
Formatting Compound Statements	12
Example: Formatting Compound Statements	13
Example: Using Braces to Clarify Flow	14
Example: Using Braces for Ease in Subsequent Modifications	15
Example: Handling Multiple Logical Expressions	16
Declaring and Initializing Variables and Data Structures	17
Using Define Statements	17
Example: #define in Source File	17
Example: #define in Header File	18
Using Typedef Statements	18
Example: Using Typedef for a User-Defined Data Structure	18

Creating Function Prototypes.....	19
Example: Creating a Business Function Prototype.....	19
Example: Creating an Internal Function Prototype.....	20
Example: Creating an External Business Function Definition.....	20
Example: Creating an Internal Function Definition.....	21
Initializing Variables	21
Example: Initializing Variables	22
Initializing Data Structures	24
Example: Using Memset to Reset the Data Structure to Null	24
Using Standard Variables	25
Flag Variables	25
Input and Output Parameters.....	25
Fetch Variables	25
Example: Using Standard Variables	26
General Coding Guidelines	28
Using Function Calls	28
Calling an External Business Function	28
Calling an Internal Business Function.....	29
Passing Pointers between Business Functions	32
Store an Address in an Array	32
Example: Storing an Address in an Array	32
Retrieve an Address from an Array.....	33
Example: Retrieving an Address from an Array.....	33
Remove an Address from an Array.....	33
Example: Removing an Address from an Array.....	33
Allocating and Releasing Memory.....	34
Example: Allocating and Releasing Memory within a Business Function	34
Using hRequest and hUser	34
Typecasting.....	35
Comparison Testing.....	35
Example: Comparison Test.....	35
Example: Creating TRUE or FALSE Test Comparison that Uses Boolean Logic.....	36
Copying Strings with jdeStrcpy versus jdeStrncpy	36
Using the Function Clean Up Area.....	37
Example: Using the Function Clean Up Area to Release Memory	37
Inserting Function Exit Points.....	37
Example: Inserting an Exit Point in a Function	38
Terminating a Function	39

Portability	40
Portability Guidelines	40
Common Server Build Errors and Warnings	41
Comments within Comments	41
New Line Character at the End of a Business Function	42
Use of Null Character	42
Lowercase Letters in Include Statements	43
Initialized Variables that are Not Referenced.....	44
EnterpriseOne Defined Structures	45
MATH_NUMERIC Data Type.....	45
JDEDATE Data Type	46
Using Malloc to Assign JDEDATE Variables	47
JDEDATECopy.....	47
Error Messages	48
Inserting Parameters in IpDS for Error Messages.....	49
Example: Parameters in IpDS for an Error Message	49
Standard Business Function Error Conditions	50
Example: Initialize Behavior Error	50
Using Text Substitution to Display Specific Error Messages.....	51
Example: Text Substitution in an Error Message.....	51
DSDE0022 Data Structure	52
Mapping Data Structure Errors with jdeCallObject.....	52
Data Dictionary Triggers	53
Example: Custom Data Dictionary Trigger	53
Unicode Compliance Standards	54
Unicode String Functions	54
Example: Using Unicode String Functions.....	55
Unicode Memory Functions	56
Example: Using jdeMemset when Setting Characters to Values other than NULL	56
Pointer Arithmetic.....	57
Offsets.....	58
MATH_NUMERIC APIs.....	58
Third-Party APIs.....	59
Example: Third-Party API.....	60
Flat-File APIs.....	60
Example: Flat-File APIs.....	61

Standard Header and Source Files	62
Standard Header	62
Business Function Name and Description	64
Copyright Notice.....	64
Header Definition for a Business Function.....	64
Table Header Inclusions	64
External Business Function Header Inclusions.....	65
Global Definitions	65
Structure Definitions.....	65
DS Template Type Definitions	65
Source Preprocessing Definitions.....	65
Business Function Prototypes.....	66
Internal Function Prototypes	66
Standard Source	66
Business Function Name and Description	69
Copyright Notice.....	69
Notes	69
Global Definitions	69
Header File for Associated Business Function	70
Business Function Header	70
Variable Declarations	70
Declare Structures	70
Pointers	70
Check for NULL Pointers	71
Set Pointers.....	71
Main Processing.....	71
Function Clean Up	71
Internal Function Comment Block.....	71
EnterpriseOne PeopleBooks Glossary	72
Index	106

About These EnterpriseOne PeopleBooks

Preface

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

This preface discusses:

- EnterpriseOne application prerequisites
- Obtaining documentation updates
- Typographical elements and visual cues
- Comments and suggestions

Note

EnterpriseOne PeopleBooks document only fields that require additional explanation. If a field 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.

EnterpriseOne Application Prerequisites

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

See the *Foundation Guide*.

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

You should be familiar with navigating the system and adding, updating, and deleting information by using EnterpriseOne menus and forms. 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 EnterpriseOne applications most effectively.

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 can find a variety of useful and timely materials, including updates to the full PeopleSoft documentation that is delivered on your PeopleBooks CD-ROM.

Note

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 Website, <http://www.peoplesoft.com/corp/en/login.jsp>

Typographical Conventions and Visual Cues

This section discusses:

- Typographical conventions
- Visual cues

Typographical Conventions

The following table contains the typographical conventions that are used in EnterpriseOne PeopleBooks:

Typographical Convention or Visual Cue	Description
<i>Italics</i>	Indicates emphasis, topic titles, and titles of PeopleSoft or other book-length publications. Also used in code to indicate variable values.
Key+Key	A plus sign (+) between keys means that you must hold down the first key while you press the second key. For example, Alt+W means hold down the Alt key while you press W.
Monospace font	Indicates a PeopleCode program or other code example.
“ ” (quotation marks)	Indicates an adjective that is used in a way that might not be readily understood without the quotation marks, for example "as of" date, "as if" currency, "from" date, and "thru" date.
Cross-references	EnterpriseOne PeopleBooks provide cross-references either below the heading "See Also" or preceded by the word See. Cross-references lead to other documentation that is pertinent to the immediately preceding documentation.

Visual Cues

EnterpriseOne PeopleBooks contain the following visual cues:

- Notes
- Cautions

Notes

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

Note

Example of a note.

Cautions

Text that is preceded by *Caution* is crucial and includes information that concerns what you must do for the system to function properly.

Caution

Example of a caution.

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 you can send e-mail comments to doc@peoplesoft.com.

While we cannot guarantee an answer to every e-mail message, we will pay careful attention to your comments and suggestions.

Business Function Programming Standards

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.

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.

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.

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.

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

Naming Conventions

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

Source and Header File Names

Source and header file names can be a maximum of 8 characters and should be formatted as follows:

bxxyyyyy

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 follows:

Ixxxxxxx_a

Where:

I = An internal function

xxxxxxx = 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 follows:

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	MATHNUMERIC
jd	JDEDATE
lp	long pointer
i	integer
by	byte
ul	unsigned long (identifier)
us	unsigned Short
ds	data structures
h	handle
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 follows:

DxxyyyyA

Where:

D = data structure

xx (second two digits) = The system code, such as:

01 - Address Book

04 - 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 follows:

D050575A

See Also

- ❑ *Creating a Business Function Data Structure* in the *Development Tools* documentation

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.

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 than 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  
}
```

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 section 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.

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.

See Also

See the following topics in the *Development Standards Business Function Programming* documentation:

- ❑ *Business Function Prototypes* for information on prototypes
- ❑ *Standard Header* for information on developing a standard header file

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_AAAAAAA( parameter list ... );

type      : Function return value
XXXXXXX   : Unique source file name
AAAAAA    : 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  
     */  
}
```

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 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);

memcpy ((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.mdbstct, &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 variable, 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:

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:

cReturnPointer	When allocating memory and returning GENLNG.
cErrorCode	Based on cCallType, cErrorCode returns a 1 when it fails or 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.

idJDEDBResult	APIs or 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. <i>Do not</i> 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

    }
}

```

General Coding Guidelines

This section contains guidelines for writing good code. It addresses using function calls, allocating and releasing memory, using hRequest and hUser, typecasting, comparison testing, copying strings with jdeStrcpy versus jdeStrncpy, inserting function exit points, and terminating 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 `CALLIBFRET` 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 call 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

```
/* 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

```
/* 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

```
/* 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

```
/* 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

```
/* 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

```
/* 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.

Store 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);

}

```

Retrieve 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);
```

Remove 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.

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 (International Organization for Standardization) 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

ERP 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.

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.

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;
```

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
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
nMonth	The month
nDay	The day

See Also

- ❑ *Messaging* in the *Development Tools* documentation for information about creating error messages and workflow messages

UsingMemcpy 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))

```

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 as follows:

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 as follows:

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);

```

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 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 `IsColumnInAddressBook`. 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;
```

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 the following:

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

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<CALENDAR_DAYS;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);
```

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.

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,
                LPDSDXXXXXXXXX  lpDS);

/*****
 * Internal Function Prototypes
 *****/
#endif      /* BXXXXXXXX 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

- ❑ *Lowercase Letters in Include Statements in the Development Standards Business Function Programming* documentation

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

- ❑ *Lowercase Letters in Include Statements* in the *Development Standards Business Function Programming* documentation

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

- ❑ *Using Define Statements* in the *Development Standards Business Function Programming* documentation

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

See the following topics in the *Development Standards Business Function Programming* documentation

- ❑ *Naming Conventions* for information on naming source and header files, functions, variables, and data structures
- ❑ *Using Typedef Statements* for information about typedef statements

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

- ❑ *Creating Function Prototypes* in the *Development Standards Business Function Programming* documentation

Internal Function Prototypes

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

See Also

See the following topics in the *Development Standards Business Function Programming* documentation:

- ❑ *Naming Conventions* for information about naming source and header files, functions, variables, and data structures
- ❑ *Creating Function Prototypes* for information about creating prototypes

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., 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.
*****/
/*****
* 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!
* LPDSDXXXXXXX lpDS Parameter Data Structure Pointer
*
*****/
JDEBFRN (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 == (LPDSDXXXXXXXX) 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

- ❑ *Using Define Statements in the Development Standards Business Function Programming documentation*

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

See the following topics in the *Development Standards Business Function Programming* documentation:

- ❑ *Naming Conventions* for information about naming source and header files, functions, variables, and data structures
- ❑ *Initializing Variables* for information about initializing variables

Declare Structures

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

See Also

- ❑ *Variable Names* in the *Development Standards Business Function Programming* documentation

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`.

See Also

- ❑ *Variable Names* in the *Development Standards Business Function Programming* documentation

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

- ❑ *Declaring and Initializing Variables and Data Structures* in the *Development Standards Business Function Programming* documentation

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

- ❑ *Using the Function Clean Up Area* in the *Development Standards Business Function Programming* documentation

Internal Function Comment Block

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

See Also

See the following topics in the *Development Standards Business Function Programming* documentation:

- ❑ *Naming Conventions* for information about naming source and header files, functions, variables, and data structures
- ❑ *Creating Function Prototypes* for information about creating prototypes

EnterpriseOne PeopleBooks Glossary

“as of” processing	A process that is run at a specific point in time to summarize item transactions.
52 period accounting	A method of accounting that uses each week as a separate accounting period.
account site	In the invoice process, the address to which invoices are mailed. Invoices can go to a different location or account site from the statement.
active window	The window that contains the document or display that will be affected by current cursor movements, commands, and data entry in environments that are capable of displaying multiple on-screen windows.
ActiveX	A technology and set of programming tools developed by Microsoft Corporation that enable software components written in different languages to interact with each another in a network environment or on a web page. The technology, based on object linking and embedding, enables Java applet-style functionality for Web browsers as well as other applications (Java is limited to Web browsers at this time). The ActiveX equivalent of a Java applet is an ActiveX control. These controls bring computational, communications, and data manipulation power to programs that can “contain” them—for example, certain Web browsers, Microsoft Office programs, and anything developed with Visual Basic or Visual C++.
activity	In Advanced Cost Accounting, an aggregation of actions performed within an organization that is used in activity-based costing.
activity driver	A measure of the frequency and intensity of the demands that are placed on activities by cost objects. An activity driver is used to assign costs to cost objects. It represents a line item on the bill of activities for a product or customer. An example is the number of part numbers, which is used to measure the consumption of material-related activities by each product, material type, or component. The number of customer orders measures the consumption of order-entry activities by each customer. Sometimes an activity driver is used as an indicator of the output of an activity, such as the number of purchase orders that are prepared by the purchasing activity. See also cost object.
activity rule	The criteria by which an object progresses from a given point to the next in a flow.
actual cost	Actual costing uses predetermined cost components, but the costs are accumulated at the time that they occur throughout the production process.
adapter	A component that connects two devices or systems, physically or electronically, and enables them to work together.
add mode	The condition of a form where a user can enter data into it.
advanced interactive executive	An open IBM operating system that is based on UNIX.
agent	A program that searches through archives or other repositories of information on a topic that is specified by the user.

aging	A classification of accounts by the time elapsed since the billing date or due date. Aging is divided into schedules or accounting periods, such as 0-30 days, 31-60 days, and so on.
aging schedule	A schedule that is used to determine whether a payment is delinquent and the number of days which the payment is delinquent.
allegato IVA clienti	In Italy, the term for the A/R Annual VAT report.
allegato IVA fornitori	In Italy, the term for the A/P Annual VAT report.
application layer	The seventh layer of the Open Systems Interconnection Reference Model, which defines standards for interaction at the user or application program level.
application programming interface (API)	A set of routines that is used by an application program to direct the performance of procedures by the computer's operating system.
AS/400 Common	A data source that resides on an AS/400 and holds data that is common to the co-existent library, allowing PeopleSoft EnterpriseOne to share information with PeopleSoft World.
assembly inclusion rule	A logic statement that specifies the conditions for using a part, adjusting the price or cost, performing a calculation, or using a routing operation for configured items.
audit trail	The detailed, verifiable history of a processed transaction. The history consists of the original documents, transaction entries, and posting of records and usually concludes with a report.
automatic return	A feature that allows a user to move to the next entry line in a detail area or to the first cell in the next row in several applications.
availability	The expression of the inventory amount that can be used for sales orders or manufacturing orders.
available inventory	The quantity of product that can be promised for sale or transfer at a particular time, considering current on-hand quantities, replenishments in process, and anticipated demand.
back office	The set of enterprise software applications that supports the internal business functions of a company.
backhaul	The return trip of a vehicle after delivering a load to a specified destination. The vehicle can be empty or the backhaul can produce less revenue than the original trip. For example, the state of Florida is considered a backhaul for many other states—that is, many trucking companies ship products into the state of Florida, but most of them cannot fill a load coming out of Florida or they charge less. Hence, trucks coming out of Florida are either empty or produce less revenue than the original trip.
balance forward	The cumulative total of inventory transactions that is used in the Running Balance program. The system does not store this total. You must run this program each time that you want to review the cumulative inventory transactions total.
balance forward receipt application method	A receipt application method in which the receipt is applied to the oldest or newest invoices in chronological order according to the net due date.

bank tape (lock box) processing	The receipt of payments directly from a customer's bank via customer tapes for automatic receipt application.
base location	[In package management] The topmost location that is displayed when a user launches the Machine Identification application.
basket discount	A reduction in price that applies to a group or "basket" of products within a sales order.
basket repricing	A rule that specifies how to calculate and display discounts for a group of products on a sales order. The system can calculate and display the discount as a separate sales order detail line, or it can discount the price of each item on a line-by-line basis within the sales order.
batch job	A job submitted to a system and processed as a single unit with no user interaction.
batch override	An instruction that causes a batch process to produce output other than what it normally would produce for the current execution only.
batch process	A type of process that runs to completion without user intervention after it has been started.
batch program	A program that executes without interacting with the user.
batch version	A version of a report or application that includes a set of user-defined specifications, which control how a batch process runs.
batch/lot tracking	The act of identifying where a component from a specific lot is used in the production of goods.
batch/mix	A manufacturing process that primarily schedules short production runs of products.
batch-of-one processing	A transaction method that allows a client application to perform work on a client workstation, and then submit the work all at once to a server application for further processing. As a batch process is running on the server, the client application can continue performing other tasks. See also direct connect, store-and-forward.
binary large object (BLOB)	A collection of binary data stored as a single entity in a [file].
binder clip	See paper clip.
black products	Products that are derived from the low or heavy end of the distillation process—for example, diesel oils and fuel oils. See also white products.
blend note	Document that authorizes a blending activity, and describes both the ingredients for the blend and the blending steps that occur.
blend off	Reworking off-specification material by introducing a small percentage back into another run of the same product.
blind execution	The mode of execution of a program that does not require the user to review or change the processing options set for the program, and does not require user intervention after the program has been launched.

boleto	In Brazil, the document requesting payment by a supplier or a bank on behalf of a supplier.
bolla doganale	VAT-Only Vouchers for Customs. In Italy, a document issued by the customs authority to charge VAT and duties on extra-EU purchasing.
bookmark	A shortcut to a location in a document or a specific place in an application or application suite.
bordero & cheque	In Brazil, bank payment reports.
broker	A program that acts as an intermediary between clients and servers to coordinate and manage requests.
BTL91	In the Netherlands, the ABN/AMRO electronic banking file format that enables batches with foreign automatic payment instructions to be delivered.
budgeted volume	A statement of planned volumes (capacity utilization) upon which budgets for the period have been set.
bunkering	A rate per ton or a sum of money that is charged for placing fuel on board; can also mean the operation itself.
business function	An encapsulated set of business rules and logic that can normally be re-used by multiple applications. Business functions can execute a transaction or a subset of a transaction (check inventory, issue work orders, and so on). Business functions also contain the APIs that allow them to be called from a form, a database trigger, or a non-EnterpriseOne application. Business functions can be combined with other business functions, forms, event rules, and other components to make up an application. Business functions can be created through event rules or third-generation languages, such as C. Examples of business functions include Credit Check and Item Availability.
business function event rule	Encapsulated, reusable business logic that is created by using through event rules rather than C programming. Contrast with embedded event rule. See also event rule.
business object library	[In interoperability] The repository that stores EnterpriseOne business objects, which consist of Java or CORBA objects.
business unit	A financial entity that is used to track the costs, revenue, or both, of an organization. A business unit can also be defined as a branch/plant in which distribution and manufacturing activities occur. Additionally, in manufacturing setup, work centers and production lines must be defined as business units; but these business unit types do not have profit/loss capability.
business view	Used by EnterpriseOne applications to access data from database tables. A business view is a means for selecting specific columns from one or more tables with data that will be used in an application or report. It does not select specific rows and does not contain any physical data. It is strictly a view through which data can be handled.
business view design aid (BDA)	An EnterpriseOne GUI tool for creating, modifying, copying, and printing business views. The tool uses a graphical user interface.

buy-back crude	In foreign producing oil countries, that portion of the host government's share of "participation crude" which it permits the company holding a concession to "buy back."
CAB	In Italy, the bank branch code or branch ID. A five-digit number that identifies any agency of a specific bank company in Italy.
cadastro de pessoas físicas	Cadastro de pessoas físicas. In Brazil, the federal tax ID for a person.
category code	A code that identifies a collection of objects sharing at least one common attribute.
central object	A software component that resides on a central server.
central objects merge	A process that blends a customer's modifications with the objects in a current release with objects in a new release.
central server	A computer that has been designated to contain the originally installed version of the software (central objects) for deployment to client computers.
certificate input	See direct input.
certificate of analysis (COA)	A document that is a record of all of the testing which has been performed against an item, lot, or both, plus the test results for that item and lot.
change management	[In software development] A process that aids in controlling and tracking the evolution of software components.
change order	In PeopleSoft, an addendum to the original purchase order that reflects changes in quantities, dates, or specifications in subcontract-based purchasing. A change order is typically accompanied by a formal notification.
chargeback	A receipt application method that generates an invoice for a disputed amount or for the difference of an unpaid receipt.
chart	EnterpriseOne term for tables of information that appear on forms in the software. See forms.
check-in location	The directory structure location for the package and its set of replicated objects. This location is usually \\deploymentserver\release\path_code\package\packagename. The subdirectories under this path are where the central C components (source, include, object, library, and DLL file) for business functions are stored.
checksum value	A computed value that depends on the contents of a block of data, and that is transmitted or stored with the data to detect whether errors have occurred in the transmission or storage.
class	[In object-oriented programming] A category of objects that share the same characteristics.
clean cargo	Term that refers to cargoes of gasoline and other refined products. See also dirty cargo.
client access	The ability to access data on a server from a client machine.
client machine	Any machine that is connected to a network and that exchanges data with a server.

client workstation	A network computer that runs user application software and is able to request data from a server.
ClieOp03	In the Netherlands, the euro-compliant uniform electronic banking file format that enables batches with domestic automatic direct debit instructions and batches with domestic payment instructions to be delivered.
ClieOp2	In the Netherlands, the uniform electronic banking file format that enables batches with domestic automatic direct debit instructions and batches with domestic payment instructions to be delivered.
cluster	Two or more computers that are grouped together in such a way that they behave like a single computer.
co-existence	A condition where two or more applications or application suites access one or more of the same database tables within the same enterprise.
cold test	The temperature at which oil becomes solid. Generally considered to be 5 degrees F lower than the pour point.
commitment	The number of items that are reserved to fill demand.
common object request broker architecture	An object request broker standard that is endorsed by the Object Management Group.
compa-ratio	An employee's salary divided by the midpoint amount for the employee's pay grade.
component changeout	See component swap.
component object model (COM)	A specification developed by Microsoft for building software components that can be assembled into programs or add functionality to existing programs running on Microsoft Windows platforms. COM components can be written in a variety of languages, although most are written in C++, and can be unplugged from a program at runtime without having to recompile the program.
component swap	In Equipment/Plant Management, the substitution of an operable component for one that requires maintenance. Typically, you swap components to minimize equipment downtime while servicing one of the components. A component swap can also mean the substitution of one parent or component item for another in its associated bill of material.
conference room pilot environment	An EnterpriseOne environment that is used as a staging environment for production data, which includes constants and masters tables such as company constants, fiscal date patterns, and item master. Use this environment along with the test environment to verify that your configuration works before you release changes to end-users.
configurable network computing (CNC)	An application architecture that allows interactive and batch applications that are composed of a single code base to run across a TCP/IP network of multiple server platforms and SQL databases. The applications consist of re-usable business functions and associated data that can be configured across the network dynamically. The overall objective for businesses is to provide a future-proof environment that enables them to change organizational structures, business processes, and technologies independently of each other.

configurable processing engine	Handles all “batch” processes, including reporting, Electronic Data Exchange (EDIt) transactions, and data duplication and transformation (for data warehousing). This ability does not mean that it exists only on the server; it can be configured to run on desktop machines (Windows 95 and NT Workstation) as well.
configuration management	A rules-based method of ordering assemble-to-order or make-to-order products in which characteristics of the product are defined as part of the Sales Order Entry process. Characteristics are edited by using Boolean logic, and then translated into the components and routing steps that are required to produce the product. The resulting configuration is also priced and costed, based on the defined characteristics.
configured item segment	A characteristic of a configured item that is defined during sales order entry. For example, a customer might specify a type of computer hard drive by stating the number of megabytes of the hard drive, rather than a part number.
consuming location	The point in the manufacturing routing where a component or subassembly is used in the production process. In kanban processing, the location where the kanban container materials are used in the manufacturing process and the kanban is checked out for replenishment.
contra/clearing account	A G/L account used by the system to offset (balance) journal entries. For example, you can use a contra/clearing account to balance the entries created by allocations.
contribution to profit	Selling price of an item minus its variable costs.
control table	A table that controls the program flow or plays a major part in program control.
control table workbench	During the Installation Workbench process, Control Table Workbench runs the batch applications for the planned merges that update the data dictionary, user defined codes, menus, and user overrides tables.
control tables merge	A process that blends a customer’s modifications to the control tables with the data that accompanies a new release.
corrective work order	A work order that is used to formally request unscheduled maintenance and communicate all of the details pertaining to the requested maintenance task.
corrective work order	A work order that is used to formally request unscheduled maintenance and communicate all of the details pertaining to the requested maintenance task.
cost assignment	Allocating resources to activities or cost objects.
cost component	An element of an item’s cost—for example, material, labor, or overhead.
cost object	Any customer, product, service, contract, project, or other work unit for which you need a separate cost measurement.
cost rollup	A simulated scenario in which work center rates, material costs, and labor costs are used to determine the total cost of an item.
costing elements	The individual classes of added value or conversion costs. These elements are typically materials, such as raw and packaging; labor and machine costs; and overhead, such as fixed and variable. Each corporation defines the necessary detail of product costs by defining and tracking cost categories and subcategories.

credit memo	A negative amount that is used to correct a customer's statement when he or she is overcharged.
credit notice	The physical document that is used to communicate the circumstances and value of a credit order.
credit order	A credit order is used to reflect products or equipment that is received or returned so that it can be viewed as a sales order with negative amounts. Credit orders usually add the product back into inventory. This process is linked with delivery confirmation.
cross segment edit	A logic statement that establishes the relationship between configured item segments. Cross segment edits are used to prevent ordering of configurations that cannot be produced.
crude oil assay	A procedure for determining the distillation curve and quality characteristics of a crude oil.
cumulative update	A version of software that includes fixes and enhancements that have been made since the last release or update.
currency relationships	When converting amounts from one currency to another, the currency relationship defines the from currency and the to currency in PeopleSoft software. For example, to convert amounts from German marks to the euro, you first define a currency relationship between those two currencies.
currency restatement	The process of converting amounts from one currency into another currency, generally for reporting purposes. It can be used, for example, when many currencies must be restated into a single currency for consolidated reporting.
current cost	The cost that is associated with an item at the time a parts list and routing are attached to a work order or rate schedule. Current cost is based on the latest bill of material and routing for the item.
customer pricing rules	In Procurement, the inventory pricing rules that are assigned to a supplier. In Sales, inventory pricing rules that are assigned to a customer.
D.A.S. 2 Reporting (DAS 2 or DADS 1)	In France, the name of the official form on which a business must declare fees and other forms of remuneration that were paid during the fiscal year.
data dictionary	A dynamic repository that is used for storing and managing a specific set of data item definitions and specifications.
data source workbench	During the Installation Workbench process, Data Source Workbench copies all of the data sources that are defined in the installation plan from the Data Source Master and Table and Data Source Sizing tables in the Planner data source to the System - release number data source. It also updates the Data Source Plan detail record to reflect completion.
data structure	A description of the format of records in a database such as the number of fields, valid data types, and so on.
data types	Supplemental information that is attached to a company or business unit. Narrative type contains free-form text. Code type contains dates, amounts, and so on.

datagram	A self-contained packet of information that is forwarded by routers, based on their address and the routing table information.
date pattern	A period of time that is set for each period in standard and 52-period accounting and forecasting.
DCE	See distributed computing environment.
DEB	See déclaration d'échange de biens.
debit memo	In Accounts Payable, a voucher that is entered with a negative amount. Enter this type of voucher when a supplier sends you a credit so that you can apply the amount to open vouchers when you issue payment to the supplier.
debit memo	A form that is issued by a customer, requesting an adjustment of the amount, which is owed to the supplier.
debit statement	A list of debit balances.
de-blend	When blend off does not result in a product that is acceptable to customers. The further processing of product to adjust specific physical and chemical properties to within specification ranges. See also blend off.
déclaration d'échange de biens (DEB)	The French term that is used for the Intrastat report.
delayed billing	The invoicing process is delayed until the end of a designated period.
delta load	A batch process that is used to compare and update records between specified environments.
denominated-in currency	The company currency in which financial reports are based.
deployment server	A server that is used to install, maintain, and distribute software to one or more enterprise servers and client workstations.
detail	The specific information that makes up a record or transaction. Contrast with summary.
detail information	Information that primarily relates to individual lines in a sales or purchase order.
direct connect	A transaction method in which a client application communicates interactively and directly with a server application. See also batch-of-one immediate, store-and-forward.
direct input	The system calculates the net units when you enter gross volume, temperature, and gravity or density. This data is generally entered during product receiving from the certificate that is prepared by an independent inspector.
direct ship orders	A purchase order that is issued to a third-party supplier who designates the destination as the customer. A direct ship sales order is also created for the customer. Direct ship orders occur when a product is not available from a company-owned or company-operated source, so the system creates an order to ship the product from a third-party source directly to the customer. Sometimes referred to as a drop ship or third-party supply.
direct usage	Consumption of resources that are attributable to specific production runs because the resources were directly issued to the schedule/order.

director	An EnterpriseOne user interface that guides a user interactively through an EnterpriseOne process.
dirty cargo	Term that refers to crude oil cargoes or other non-refined petroleum cargoes. See also clean cargo.
dispatch planning	Efficient planning and scheduling of product deliveries. Considerations include: Dispatch groups Scheduled delivery date Scheduled delivery time Preferred delivery date Preferred delivery time Average delivery time for that geographical location Available resources Special equipment requirements at the product's source or destination.
displacement days	The number of days that are calculated from today's date by which you group vouchers for payment. For example, if today's date is March 10 and you specify three displacement days, the system includes vouchers with a due date through March 13 in the payment group. Contrast with pay-through date.
display sequence	A number that the system uses to re-order a group of records on the form.
distributed computing environment (DCE)	A set of integrated software services that allows software which is running on multiple computers to perform seamless and transparently to the end-users. DCE provides security, directory, time, remote procedure calls, and files across computers running on a network.
distributed data processing	Processing in which some of the functions are performed across two or more linked facilities or systems.
distributed database management system (DDBMS)	A system for distributing a database and its control system across many geographically dispersed machines.
do not translate (DNT)	A type of data source that must exist on the AS/400 because of BLOB restrictions.
double-byte character set (DBCS)	A method of representing some characters by using one byte and other characters by using two bytes. Double-byte character sets are necessary to represent some characters in the Japanese, Korean, and Chinese languages.
downgrade profile	A statement of the hierarchy of allowable downgrades. Includes substitutions of items, and meeting tighter specifications for those products with wider or overlapping specification ranges.
DTA	Datenträgeraustausch. A Swiss payment format that is required by Telekurs (Payserv).
dual pricing	To provide prices for goods and services in two currencies. During the euro transition period, dual pricing between the euro and Economic and Monetary Union (EMU) member currencies is encouraged.

dynamic link library (DLL)	A set of program modules that are designed to be invoked from executable files when the executable files are run, without having to be linked to the executable files. They typically contain commonly used functions.
dynamic partitioning	The ability to dynamically distribute logic or data to multiple tiers in a client/server architecture.
economy of scale	A phenomenon whereby larger volumes of production reduce unit cost by distributing fixed costs over a larger quantity. Variable costs are constant; but fixed costs per unit are reduced, thereby reducing total unit cost.
edit mode	A processing mode or condition where the user can alter the information in a form.
edit rule	A method that is used for formatting user entries, validating user entries, or both, against a predefined rule or set of rules.
embedded event rule	An event rule that is specific to a particular table or application. Examples include form-to-form calls, hiding a field that is based on a processing option value, or calling a business function. Contrast with business function event rule. See also event rule.
employee work center	A central location for sending and receiving all EnterpriseOne messages (system and user-generated), regardless of the originating application or user. Each user has a mailbox that contains workflow and other messages, including Active Messages. With respect to workflow, the Message Center is MAPI compliant and supports drag-and-drop work reassignment, escalation, forward and reply, and workflow monitoring. All messages from the message center can be viewed through EnterpriseOne messages or Microsoft Exchange.
Emulator	An item of software or firmware that allows one device to imitate the functioning of another.
encapsulation	The ability to confine access to and manipulation of data within an object to the procedures that contribute to the definition of that object.
engineering change order (ECO)	A work order document that is used to implement and track changes to items and resulting assemblies. The document can include changes in design, quantity of items required, and the assembly or production process.
enhanced analysis database	A database containing a subset of operational data. The data on the enhanced analysis database performs calculations and provides summary data to speed generation of reports and query response times. This solution is appropriate when external data must be added to source data, or when historical data is necessary for trend analysis or regulatory reporting. See also duplicated database, enterprise data warehouse.
enterprise server	A computer containing programs that collectively serve the needs of an enterprise rather than a single user, department, or specialized application.
EnterpriseOne object	A re-usable piece of code that is used to build applications. Object types include tables, forms, business functions, data dictionary items, batch processes, business views, event rules, versions, data structures, and media objects. See also object.

EnterpriseOne process	Allows EnterpriseOne clients and servers to handle processing requests and execute transactions. A client runs one process, and servers can have multiple instances of a process. EnterpriseOne processes can also be dedicated to specific tasks (for example, workflow messages and data replication) to ensure that critical processes do not have to wait if the server is particularly busy.
EnterpriseOne web development computer	A standard EnterpriseOne Windows developer computer with the additional components installed: Sun's JDK 1.1. JFC (0.5.1). Generator Package with Generator.Java and JDECOM.dll. R2 with interpretive and application controls/form.
environment workbench	During the Installation Workbench process, Environment Workbench copies the environment information and Object Configuration Manager tables for each environment from the Planner data source to the System release number data source. It also updates the Environment Plan detail record to reflect completion.
equivalent fuel	A barrel of equivalent fuel supplies six million BTUs of heat. Fuel gas quantities are usually calculated as equivalent fuel barrels in economic calculations for refinery operations.
escalation monitor	A batch process that monitors pending requests or activities, and restarts or forwards them to the next step or user after they have been inactive for a specified amount of time.
ESR	Einzahlungsschein mit Referenznummer. A pay slip with a reference number.
event rule	[In EnterpriseOne] A logic statement that instructs the system to perform one or more operations that are based on an activity that can occur in a specific application, such as entering a form or exiting a field.
exit bar	[In EnterpriseOne] The tall pane with icons in the left portion of many EnterpriseOne program windows.
facility	An entity within a business for which you want to track costs. For example, a facility might be a warehouse location, job, project, work center, or branch/plant. Sometimes referred to as a business unit.
fast path	[In EnterpriseOne] A command prompt that allows the user to move quickly among menus and applications by using specific commands.
file handle	A temporary reference (typically a number) that is assigned to a file which has been opened by the operating system and is used throughout the session to access the file.
file server	A computer that stores files to be accessed by other computers on the network.
find/browse	A type of form used to: Search, view, and select multiple records in a detail area. Delete records. Exit to another form. Serve as an entry point for most applications.

firm planned order (FPO)	A work order that has reached a user defined status. When this status is entered in the processing options for the various manufacturing programs, messages for those orders are not exploded to the components.
fiscal date pattern	A representation of the beginning date for the fiscal year and the ending date for each period in that year.
fix/inspect	A type of form used to view, add, or modify existing records. A fix/inspect form has no detail area.
fixed quantity	A term that indicates the bill of material relationship between a parent item and its components or ingredients. When a bill of material component has a fixed quantity relationship to its parent, the amount of the component does not change when the software calculates parts list requirements for different work order quantities. Contrast with variable quantity.
flexible account numbers	The format of account numbers for journal entries. The format that you set up must be the three segments: Business unit. Object. Subsidiary.
form design aid (FDA)	The EnterpriseOne GUI development tool for building interactive applications and forms.
form exit	[In EnterpriseOne] An option that is available as a button on the Form Exit bar or as a selection in the Form menu. It allows users to open an interconnected form.
form interconnection	Allows one form to access and pass data to another form. Form interconnections can be attached to any event; however, they are normally used when a button is clicked.
form type	The following form types are available in EnterpriseOne: Find/browse. Fix/inspect. Header detail. Headerless detail. Message. Parent/child. Search/select.
form-to-form call	A request by a form for data or functionality from one of the connected forms.
framework	[In object-oriented systems] A set of object classes that provide a collection of related functions for a user or piece of software.
frozen cost	The cost of an item, operation, or process after the frozen update program is run; used by the Manufacturing Accounting system.
frozen update program	A program that freezes the current simulated costs, thereby finalizing them for use by the Manufacturing Accounting system.

globally unique identifier (GUI)	A 16-byte code in the Component Object Model that identifies an interface to an object across all computers and networks.
handle	[In programming] A pointer that contains the address of another pointer, which, in turn, contains the address of the desired object.
hard commitment	The number of items that are reserved for a sales order, work order, or both, from a specific location, lot, or both.
hard error	An error that cannot be corrected by a given error detection and correction system.
header	Information at the beginning of a table or form. Header information is used to identify or provide control information for the group of records that follows.
header information	Information that pertains to the entire order.
hover help	A help function that provides contextual information or instructions when a cursor moves over a particular part of the interface element for a predefined amount of time.
ICMS	Imposto sobre circulação de mercadoria e serviços. In Brazil, a state tax that is applied to the movement of merchandise and some services.
ICMS Substituto	Imposto sobre circulação de mercadoria e serviços substituto. In Brazil, the ICMS tax that is charged on interstate transactions, or on special products and clients.
ICMS Substituto-Markup	See imposto sobre circulação de mercadoria e serviços substituto-markup.
imposto de renda (IR)	Brazilian income tax.
imposto sobre produtos industrializados	In Brazil, a federal tax that applies to manufactured goods (domestic and imported).
imposto sobre services (ISS)	In Brazil, tax on services.
inbound document	A document that is received from a trading partner using Electronic Data Interface (EDI). This document is also referred to as an inbound transaction.
indented tracing	Tracking all lot numbers of intermediates and ingredients that are consumed in the manufacture of a given lot of product, down through all levels of the bill of material, recipe, or formula.
indexed allocations	A procedure that allocates or distributes expenses, budgets, adjustments, and so on, among business units, based on a fixed percentage.
indirect measurement	Determining the quantity on-hand by: Measuring the storage vessels and calculating the content's balance quantity. or Theoretically calculating consumption of ingredients and deducting them from the on-hand balance.

indirect usage	Determining what should have been used by multiplying receipt quantity of the parent times the quantity per statement in the formula, recipe, or bill of material. This transaction typically affects both consumption on schedule as well as issue from on-hand balances.
in-process rework	Recycling a semi processed product that does not meet acceptable standards. Further processing takes the product out of a given operation and sends it back to the beginning of that operation or a previous operation (for example, unreacted materials). Rework that is detected prior to receipt of finished goods and corrected during the same schedule run.
INPS withholding tax	Instituto Nazionale di Previdenza Sociale withholding tax. In Italy, a 12% social security withholding tax that is imposed on payments to certain types of contractors. This tax is paid directly to the Italian social security office.
inscrição estadual	ICMS tax ID. In Brazil, the state tax ID.
inscrição municipal	ISS tax ID. In Brazil, the municipal tax ID.
integrated toolset	Unique to EnterpriseOne is an industrial-strength toolset that is embedded in the already comprehensive business applications. This toolset is the same toolset that is used by PeopleSoft to build EnterpriseOne interactive and batch applications. Much more than a development environment, however, the EnterpriseOne integrated toolset handles reporting and other batch processes, change management, and basic data warehousing facilities.
integrity test	A process that is used to supplement a company's internal balancing procedures by locating and reporting balancing problems and data inconsistencies.
interbranch sales order	A sales order that is used for transactions between branch/plants other than the selling branch/plant.
Interoperability	The ability of different computer systems, networks, operating systems, and applications to work together and share information.
inventory pricing rule	A discount method that is used for purchases from suppliers and sales to customers. The method is based on effectivity dates, up-to quantities, and a factor by which you can mark up or discount the price or cost.
inventory turn	The number of times that the inventory cycles, or turns over, during the year. A frequently used method to compute inventory turnover is to divide the annual costs of sales by the average inventory level.
invoice	An itemized list of goods that are shipped or services that are rendered, stating quantities, prices, fees, shipping charges, and so on. Companies often have their invoices mailed to a different address than where they ship products. In such cases, the bill-to address differs from the ship-to address.
IP	See imposto sobre produtos industrializados.
IR	See imposto de renda.
IServer Service	Developed by PeopleSoft, this Internet server service resides on the Web server and is used to speed up delivery of the Java class files from the database to the client.

ISS	See imposto sobre servicios.
jargon	An alternate data dictionary item description that EnterpriseOne or PeopleSoft World displays, based on the product code of the current object.
java application server	A component-based server that resides in the middle-tier of a server-centric architecture and provides middleware services for security and state maintenance, along with data access and persistence.
JDBNET	A database driver that allows heterogeneous servers to access each other's data.
jde.ini	A PeopleSoft file (or member for AS/400) that provides the runtime settings that are required for EnterpriseOne initialization. Specific versions of the file or member must reside on every machine that is running EnterpriseOne, including workstations and servers.
JDE.LOG	The main diagnostic log file of EnterpriseOne. Always located in the root directory on the primary drive. Contains status and error messages from the startup and operation of EnterpriseOne.
JDEBASE Database Middleware	PeopleSoft proprietary database middleware package that provides two primary benefits: 1. Platform-independent APIs for multidatabase access. These APIs are used in two ways: a. By the interactive and batch engines to dynamically generate platform-specific SQL, depending on the data source request. b. As open APIs for advanced C business function writing. These APIs are then used by the engines to dynamically generate platform-specific SQL. 2. Client-to-server and server-to-server database access. To accomplish this access, EnterpriseOne is integrated with a variety of third-party database drivers, such as Client Access 400 and open database connectivity (ODBC).
JDECallObject	An application programming interface that is used by business functions to invoke other business functions.
JDEIPC	Communications programming tools that are used by server code to regulate access to the same data in multiprocess environments, communicate and coordinate between processes, and create new processes.
JDENET	PeopleSoft proprietary middleware software. JDENET is a messaging software package.
JDENET communications middleware	PeopleSoft proprietary communications middleware package for EnterpriseOne. It is a peer-to-peer, message-based, socket-based, multiprocess communications middleware solution. It handles client-to-server and server-to-server communications for all EnterpriseOne supported platforms.
just in time installation (JITI)	EnterpriseOne's method of dynamically replicating objects from the central object location to a workstation.
just in time replication (JITR)	EnterpriseOne's method of replicating data to individual workstations. EnterpriseOne replicates new records (inserts) only at the time that the user needs the data. Changes, deletes, and updates must be replicated using Pull Replication.

Kagami	In Japan, summarized invoices that are created monthly (in most cases) to reduce the number of payment transactions.
latitude	The X coordinate of the location of an item in the warehouse. The system can use latitude, longitude, and height when suggesting locations for putaway, replenishment, and picking.
laytime (or layhours)	The amount of time that is allotted to a tanker at berth to complete loading or discharging cargo. This time is usually expressed in running hours, and is fixed by prior agreement between the vessel owner and the company that is chartering the vessel. Laytime is stipulated in the charter, which states exactly the total of number of hours that are granted at both loading and unloading ports, and indicates whether such time is reversible. A statement of “Seventy-Two Hours, Reversible” means that a total of 72 hours is granted overall at both ports, and any time saved at one port can be applied as a credit at the other port. For example, if the vessel uses only 32 hours instead of 36 hours to load cargo, it can apply an additional four hours to the 36 hours allotted at the discharge port. Such considerations are important for purposes of computing demurrage.
leading zeros	A series of zeros that certain facilities in PeopleSoft systems place in front of a value that is entered. This situation normally occurs when you enter a value that is smaller than the specified length of the field. For example, if you enter 4567 in a field that accommodates eight numbers, the facility places four zeros in front of the four numbers that you enter. The result appears as 00004567.
ledger type	A code that designates a ledger which is used by the system for a particular purpose. For example, all transactions are recorded in the AA (actual amounts) ledger type in their domestic currency. The same transactions can also be stored in the CA (foreign currency) ledger type.
level break	The position in a report or text where a group of similar types of information ends and another one begins.
libro IVA	Monthly VAT report. In Italy, the term for the report that contains the detail of invoices and vouchers that were registered during each month.
line of business	A description of the nature of a company’s work; also a tool to control the relationship with that customer, including product pricing.
linked service type	A service type that is associated with a primary service type. Linked service types can be cancelled, and the maintenance tasks are performed when the primary service type to which they are linked comes due. You can specify whether the system generates work orders for linked service types, as well as the status that the system assigns to work orders that have already been generated. Sometimes referred to as associated service types. See also primary service type and service type.
livro razao	In Brazil, a general ledger report.
load balancing	The act of distributing the number of processes proportionally to all servers in a group to maximize overall performance.
location workbench	During the Installation Workbench process, Location Workbench copies all locations that are defined in the installation plan from the Location Master table in the Planner data source to the System data source.

log files	Files that track operations for a process or application. Reviewing log files is helpful for troubleshooting problems. The file extension for log files is .LOG.
logic data source	Any code that provides data during runtime.
logical compartment	One of two ways that is identified in the transportation constants to display compartments on vehicles. Logical display numbers the compartments sequentially. For example, if two vehicles are on a trip and each vehicle has three compartments, the logical display is 1,2,3,4,5,6.
logical file	A set of keys or indices that is used for direct access or ordered access to the records in a physical file. Several logical files can have different accesses to a physical.
logical shelf	A logical, not physical, location for inventory that is used to track inventory transactions in loan/borrow, or exchange agreements with other companies. See also logical warehouse.
logical warehouse	Not a physical warehouse containing actual inventory, but a means for storing and tracking information for inventory transactions in loan/borrow, or exchange agreements with other companies.
longitude	The Y coordinate of the location of an item in the warehouse. The system can use latitude, longitude, and height when suggesting locations for putaway, replenishment, and picking.
LSV	Lastschriftverfahren. A Swiss auto debit format that is required by Telekurs (Payserv).
mail merge	A mass-mail facility that takes names, addresses, and (sometimes) pertinent facts about recipients and merges the information into a form letter or a similarly basic document.
mailmerge workbench	[In EnterpriseOne] An application that merges Microsoft Word 6.0 (or higher) word-processing documents with EnterpriseOne records to automatically print business documents.
main fuels	Usually refers to bulk fuel products, but sometimes includes packaged products.
maintenance loop	See maintenance route.
maintenance route	A method of performing PMs for multiple pieces of equipment from a single preventive maintenance work order. A maintenance route includes pieces of equipment that share one or more identical maintenance tasks which can be performed at the same time for each piece of equipment. Sometimes referred to as maintenance loop.
maintenance work order	In PeopleSoft EnterpriseOne systems, a term that is used to distinguish work orders created for the performance of equipment and plant maintenance from other work orders, such as manufacturing work orders, utility work orders, and engineering change orders.

manufacturing and distribution planning	Planning that includes resource and capacity planning, and material planning operations. Resource and capacity planning allows you to prepare a feasible production schedule that reflects your demand forecasts and production capability. Material Planning Operations provides a short-range plan to cover material requirements that are needed to make a product.
mapping	A set of instructions that describes how one data structure passes data to another.
master business function	An interactive master file that serves as a central location for adding, changing, and updating information in a database.
master business function	A central system location for standard business rules about entering documents, such as vouchers, invoices, and journal entries. Master business functions ensure uniform processing according to guidelines that you establish.
master table	A database table that is used to store data and information that is permanent and necessary to the system's operation. Master tables might contain data such as paid tax amounts, supplier names, addresses, employee information, and job information.
matching document	A document that is associated with an original document to complete or change a transaction. For example, a receipt is the matching document of an invoice.
media object	An electronic or digital representation of an object.
media storage objects	Files that use one of the following naming conventions that are not organized into table format: Gxxx, xxxGT, or GTxxx.
memory violation	An error that occurs as the result of a memory leak.
menu selection	An option on a menu that initiates a software function directly.
message center	A central location for sending and receiving all EnterpriseOne messages (system- and user-generated), regardless of the originating application or user.
messaging application programming interface (MAPI)	An architecture that defines the components of a messaging system and how they behave. It also defines the interface between the messaging system and the components.
metal content	A series of properties of a blended product that help to determine its suitability for a prescribed purpose.
metals management	The process of maintaining information about the location and status of durable product containers such as liquid petroleum gas (LPG) cylinders.
mobile inventory	Inventory that is transferred from a depot to a barge or truck for milk-run deliveries.
modal	A restrictive or limiting interaction that is created by a given condition of operation. Modal often describes a secondary window that restricts a user's interaction with other windows. A secondary window can be modal with respect to its primary window or to the entire system. A modal dialog box must be closed by the user before the application continues.

model work order	For scheduled preventive maintenance or for a condition-based alert, a model work order functions as a template for the creation of other work orders. You can assign model work orders to service types and condition-based alerts. When the service type comes due or the alert is generated, the system automatically generates a work order that is based on information from the model work order.
modeless	Not restricting or limiting interaction. Modeless often describes a secondary window that does not restrict a user's interaction with other windows. A modeless dialog box stays on the screen and is available for use at any time, but also permits other user activities.
multiple stocking locations	Authorized storage locations for the same item number at locations, in addition to the primary stocking location.
multitier architecture	A client/server architecture that allows multiple levels of processing. A tier defines the number of computers that can be used to complete some defined task.
named event rules (NER)	Also called business function event rules. Encapsulated, re-usable business logic that is created by using event rules, rather than C programming.
national language support (NLS)	Mechanisms that are provided to facilitate internationalization of both system and application user interfaces.
natureza da operação	Transaction nature. In Brazil, a code that classifies the type of commercial transaction to conform to the fiscal legislation.
negative pay item	An entry in an account that indicates a prepayment. For example, you might prepay a supplier before goods are sent or prepay an employee's forecasted expenses for a business trip. The system stores these pending entries, assigning them a minus quantity as debit amounts in a designated expense account. After the prepaid goods are received or the employee submits an expense report, entering the actual voucher clears all of the negative pay items by processing them as regular pay items. Note that a negative pay item can also result from entering a debit memo (A/P) or a credit memo (A/R).
net added cost	The cost to manufacture an item at the current level in the bill of material. Thus, for manufactured parts, the net added cost includes labor, outside operations, and cost extras applicable to this level in the bill of material, but not materials (lower-level items). For purchased parts, the net added cost also includes the cost of materials.
next status	The next step in the payment process for payment control groups. The next status can be either WRT (write) or UPD (update).
node	A termination point for two or more communications links. A node can serve as the control location for forwarding data among the elements of a network or multiple networks, as well as performing other networking and, in some cases, local processing.
non-inventory items	See non-stock items.
non-list price	A price for bulk products that is determined by its own algorithms, such as a rolling average or commodity price plus.
non-prime product	A manufactured product with revenue potential that is less than the product planned for, or scheduled to be produced.

non-stock items	Items that the system does not account for as part of the inventory. For example, office supplies, or packaging materials can be non-stock items.
nota fiscal	In Brazil, a legal document that must accompany all commercial transactions.
nota fiscal fatura	In Brazil, a nota fiscal and invoice information.
notula	In Italy, the process whereby a business does not recognize value added tax until the payment of a voucher.
object configuration manager (OCM)	EnterpriseOne's object request broker and the control center for the runtime environment. It keeps track of the runtime locations for business functions, data, and batch applications. When one of these objects is called, the Object Configuration Manager directs access to it by using defaults and overrides for a given environment and user.
object embedding	When an object is embedded in another document, an association is maintained between the object and the application that created it; however, any changes made to the object are also only kept in the compound document. See also object linking.
object librarian	A repository of all versions, applications, and business functions that are re-usable in building applications.
object linking	When an object is linked to another document, a reference is created with the file in which the object is stored, as well as with the application that created it. When the object is modified, either from the compound document or directly through the file in which it is saved, the change is reflected in that application as well as anywhere it has been linked. See also object embedding.
object linking and embedding (OLE)	A technology for transferring and sharing information among applications by allowing the integration of objects from diverse applications, such as graphics, charts, spreadsheets, text, or an audio clip from a sound program. OLE is a compound document standard that was developed by Microsoft Corporation. It enables you to create objects with one application, and then link or embed them in a second application. Embedded objects retain their original format and links to the application that created them. See also object embedding, object linking.
object management workbench (OMW)	The change management system that is used for EnterpriseOne development.
object-based technology (OBT)	A technology that supports some of the main principles of object-oriented technology: Classes. Polymorphism. Inheritance. Encapsulation.

object-oriented technology (OOT)	Brings software development past procedural programming into a world of reusable programming that simplifies development of applications. Object orientation is based on the following principles: Classes. Polymorphism.I Inheritance. Encapsulation.
offsetting account	An account that reduces the amount of another account to provide a net balance. For example, a credit of 200 to a cash account might have an offsetting entry of 200 to an A/P Trade (liability) account.
open database connectivity (ODBC)	Defines a standard interface for different technologies to process data between applications and different data sources. The ODBC interface comprises set of function calls, methods of connectivity, and representation of data types that define access to data sources.
open systems interconnection (OSI)	The OSI model was developed by the International Standards Organization (ISO) in the early 1980s. It defines protocols and standards for the interconnection of computers and network equipment.
order detail line	A part of an order that contains transaction information about a service or item being purchased or sold, such as quantity, cost, price, and so on.
order hold	A flag that stops the processing of an order because it has exceeded the credit or budget limit, or has another problem.
order-based pricing	Pricing strategy that grants reductions in price to a customer. It is based upon the contents and relative size (volume or value) of the order as a whole.
outbound document	A document that is sent to a trading partner using EDI. This term is also referred to as an outbound transaction.
outturn	The quantity of oil that is actually received into a buyer's storage tanks when a vessel is unloaded. For various reasons (vaporization, clingage to vessel tank walls, and so on), the amount of a product pumped into shore tankage at unloading is often less than the quantity originally loaded onto the vessel, as certified by the Bill of Lading. Under a delivered or CIF outturn transaction, the buyer pays only for the barrels actually "turned out" by the vessel into storage. When a buyer is paying CIF Bill of Lading figures, a loss of 0.5% of total cargo volume is considered normal. Losses in excess of 0.5%, however, are either chargeable to the seller or are covered by specialized insurance that covers partial, as well as total, loss of the cargo.
overhead	In the distillation process, that portion of the charge that leaves the top of the distillation column as vapor. This definition is strictly as it relates to ECS.
override conversion method	A method of calculating exchange rates that is set up between two specific currencies. For those specific currencies, this method overrides the conversion method in General Accounting Constants and does not allow inverse rates to be used when calculating currency amounts.

package / package build	A collection of software that is grouped into a single entity for modular installation. EnterpriseOne objects are installed to workstations in packages from the deployment server. A package can be compared to a bill of material or kit that indicates the necessary objects for that workstation and where the installation program can find them on the deployment server. It is a point-in-time “snapshot” of the central objects on the deployment server.
package location	The directory structure location for the package and its set of replicated objects. This location is usually \\deployment server\release\path_code\package\ package name. The replicated objects for the package are placed in the subdirectories under this path. This location is also where the package is built or stored.
package workbench	During the Installation Workbench process, Package Workbench transfers the package information tables from the Planner data source to the System - release number data source. It also updates the Package Plan detail record to reflect completion.
packaged products	Products that, by their nature, must be delivered to the customer in containers which are suitable for discrete consumption or resale.
pane/panel	A resizable subarea of a window that contains options, components, or other related information.
paper clip	An icon that is used to indicate that a media object is attached to a form or record.
parent/child form	A type of form that presents parent/child relationships in an application on one form: The left portion of the form presents a tree view that displays a visual representation of a parent/child relationship. The right portion of the form displays a detail area in browse mode. The detail area displays the records for the child item in the tree. The parent/child form supports drag and drop functionality.
parent/child relationship	See parent/component relationship.
parent/component relationship	1. In Capital Asset Management, the hierarchical relationship of a parent piece of equipment to its components. For example, a manufacturing line could be a parent and the machinery on the line could be components of the line. In addition, each piece of machinery could be a parent of still more components. 2. In Product Data Management, a hierarchical relationship of the components and subassemblies of a parent item to that parent item. For example, an automobile is a parent item; its components and subassemblies include: engine, frame, seats, and windows. Sometimes referred to as parent/child relationship.
partita IVA	In Italy, a company fiscal identification number.
pass-through	A process where data is accepted from a source and forwarded directly to a target without the system or application performing any data conversion, validation, and so on.
pay on consumption	The method of postponing financial liability for component materials until you issue that material to its consuming work order or rate schedule.

payment group	A system-generated group of payments with similar information, such as a bank account. The system processes all of the payments in a payment group at the same time.
PeopleSoft database	See JDEBASE Database Middleware.
performance tuning	The adjustments that are made for a more efficient, reliable, and fast program.
persistent object	An object that continues to exist and retains its data beyond the duration of the process that creates it.
pervasive device	A type of intelligent and portable device that provides a user with the ability to receive and gather information anytime, from anywhere.
planning family	A means of grouping end items that have similarity of design or manufacture.
plug-in	A small program that plugs into a larger application to provide added functionality or enhance the main application.
polymorphism	A principle of object-oriented technology in which a single mnemonic name can be used to perform similar operations on software objects of different types.
portal	A Web site or service that is a starting point and frequent gateway to a broad array of on-line resources and services.
Postfinance	A subsidiary of the Swiss postal service. Postfinance provides some banking services.
potency	Identifies the percent of an item in a given solution. For example, you can use an 80% potent solution in a work order that calls for 100% potent solution, but you would use 25% more, in terms of quantity, to meet the requirement ($100 / 80 = 1.25$).
preference profile	The ability to define default values for specified fields for a user defined hierarchy of items, item groups, customers, and customer groups. In Quality Management setup, this method links test and specification testing criteria to specific items, item groups, customers, or customer groups.
preflush	A work order inventory technique in which you deduct (relieve) materials from inventory when the parts list is attached to the work order or rate schedule.
preventive maintenance cycle	The sequence of events that make up a preventive maintenance task, from its definition to its completion. Because most preventive maintenance tasks are commonly performed at scheduled intervals, parts of the preventive maintenance cycle repeat, based on those intervals.
preventive maintenance schedule	The combination of service types that apply to a specific piece of equipment, as well as the intervals at which each service type is scheduled to be performed.
primary service type	A service type to which you can link related service types. For example, for a particular piece of equipment, you might set up a primary service type for a 1000-hour inspection and a linked service type for a 500-hour inspection. The 1000-hour inspection includes all of the tasks performed at 500 hours. When a primary service type is scheduled to be performed, the system schedules the linked service type. See also linked service type.

pristine environment	An EnterpriseOne environment that is used to test unaltered objects with PeopleSoft demonstration data or for training classes. You must have this environment so you can compare pristine objects that you modify.
processing option	A data structure that allows users to supply parameters that regulate the execution of a batch program or report.
product data management (PDM)	In PeopleSoft EnterpriseOne software, the system that enables a business to organize and maintain information about each item which it manufactures. Features of this system, such as bills of material, work centers, and routings, define the relationships among parents and components, and how they can be combined to manufacture an item. PDM also provides data for other manufacturing systems including Manufacturing Accounting, Shop Floor Management, and Manufacturing and Distribution Planning.
product line	A group of products with similarity in manufacturing procedures, marketing characteristics, or specifications that allow them to be aggregated for planning; marketing; and, occasionally, costing.
product/process definition	A combination of bill of material (recipe, formula, or both) and routing (process list). Organized into tasks with a statement of required consumed resources and produced resources.
production environment	An EnterpriseOne environment in which users operate EnterpriseOne software.
program temporary fix (PTF)	A representation of changes to PeopleSoft software that your organization receives on magnetic tapes or diskettes.
project	[In EnterpriseOne] A virtual container for objects being developed in Object Management Workbench.
projected cost	The target expenditure in added value for material, labor, and so on, during manufacture. See also standard cost.
promotion path	The designated path for advancing objects or projects in a workflow.
protocollo	See registration number.
PST	Provincial sales tax. A tax that is assessed by individual provinces in Canada.
published table	Also called a “Master” table, this is the central copy to be replicated to other machines and resides on the “publisher” machine. The Data Replication Publisher Table (F98DRPUB) identifies all of the published tables and their associated publishers in the enterprise.
publisher	The server that is responsible for the published table. The Data Replication Publisher Table (F98DRPUB) identifies all of the published tables and their associated publishers in the enterprise.
pull replication	One of the EnterpriseOne methods for replicating data to individual workstations. Such machines are set up as pull subscribers that use EnterpriseOne’s data replication tools. The only time that pull subscribers are notified of changes, updates, and deletions is when they request such information. The request is in the form of a message that is sent, usually at startup, from the pull subscriber to the server machine that stores the Data Replication Pending Change Notification table (F98DRPCN).

query by example (QBE)	Located at the top of a detail area, this area is used to search for data to display in the detail area.
rate scheduling	A method of scheduling product or manufacturing families, or both. Also a technique to determine run times and quantities of each item within the family to produce enough of each individual product to satisfy demand until the family can be scheduled again.
rate type	For currency exchange transactions, the rate type distinguishes different types of exchange rates. For example, you can use both period average and period-end rates, distinguishing them by rate type.
real-time	Pertaining to information processing that returns a result so rapidly that the interaction appears to be instantaneous.
receipt routing	A series of steps that is used to track and move items within the receipt process. The steps might include in-transit, dock, staging area, inspection, and stock.
referential integrity	Ensures that a parent record cannot be deleted from the database when a child record for exists.
regenerable	Source code for EnterpriseOne business functions can be regenerated from specifications (business function names). Regeneration occurs whenever an application is recompiled, either for a new platform or when new functionality is added.
register types and classes	In Italian VAT Summary Reporting, the classification of VAT transactions.
relationship	Links tables together and facilitates joining business views for use in an application or report. Relationships that are created are based on indexes.
relevé d'identité bancaire (RIB)	In France, the term that indicates the bank transit code, account number, and check digit that are used to validate the bank transit code and account number. The bank transit code consists of the bank code and agency code. The account number is alphanumeric and can be as many as 11 characters. PeopleSoft supplies a validation routine to ensure RIB key correctness.
remessa	In Brazil, the remit process for A/R.
render	To include external data in displayed content through a linking mechanism.
repasse	In Brazil, a discount of the ICMS tax for interstate transactions. It is the adjustment between the interstate and the intrastate ICMS tax rates.
replenishment point	The location on or near the production line where additional components or subassemblies are to be delivered.
replication server	A server that is responsible for replicating central objects to client machines.
report design aid (RDA)	The EnterpriseOne GUI tool for operating, modifying, and copying report batch applications.
repost	In Sales, the process of clearing all commitments from locations and restoring commitments, based on quantities from the Sales Order Detail table (F4211).
resident	Pertaining to computer programs or data while they remain on a particular storage device.

retorno	In Brazil, the receipt process for A/R.
RIB	See relevé d'identité bancaire.
ricevute bancarie (RiBa)	In Italy, the term for accounts receivable drafts.
riepilogo IVA	Summary VAT monthly report. In Italy, the term for the report that shows the total amount of VAT credit and debit.
ritenuta d'acconto	In Italy, the term for standard withholding tax.
rollback	[In database management] A feature or command that undoes changes in database transactions of one or more records.
rollup	See cost rollup.
row exit	[In EnterpriseOne] An application shortcut, available as a button on the Row Exit bar or as a menu selection, that allows users to open a form that is related to the highlighted grid record.
runtime	The period of time when a program or process is running.
SAD	The German name for a Swiss payment format that is accepted by Postfinance.
SAR	See software action request.
scalability	The ability of software, architecture, hardware, or a network to support software as it grows in size or resource requirements.
scripts	A collection of SQL statements that perform a specific task.
scrub	To remove unnecessary or unwanted characters from a string.
search/select	A type of form that is used to search for a value and return it to the calling field.
selection	Found on PeopleSoft menus, selections represent functions that you can access from a menu. To make a selection, type the associated number in the Selection field and press Enter.
serialize	To convert a software object into a stream of bytes to store on a disk or transfer across a network.
server map	The server view of the object configuration mapping.
server workbench	During the Installation Workbench process, Server Workbench copies the server configuration files from the Planner data source to the System release number data source. It also updates the Server Plan detail record to reflect completion.
service interval	The frequency at which a service type is to be performed. Service intervals can be based on dates, periods, or statistical units that are user defined. Examples of statistical units are hours, miles, and fuel consumption.
service type	An individual preventive maintenance task or procedure, such as an inspection, lubrication, or overhaul. Service types can apply to a specific piece of equipment or to a class of equipment. You can specify that service types come due based on a predetermined service interval, or whenever the task that is represented by the service type becomes necessary.

servlet	A [small] program that extends the functionality of a Web server by generating dynamic content and interacting with Web clients by using a request-response paradigm.
share path	The network node under which one or more servers or objects reside.
shop floor management	A system that uses data from multiple system codes to help develop, execute, and manage work orders and rate schedules in the enterprise.
silent mode	A method for installing or running a program that does not require any user intervention.
silent post	A type of post that occurs in the background without the knowledge of the user.
simulated cost	After a cost rollup, the cost of an item, operation, or process according to the current cost scenario. This cost can be finalized by running the frozen update program. You can create simulated costs for a number of cost methods—for example, standard, future, and simulated current costs. See also cost rollup.
single-byte character set (SBCS)	An encoding scheme in which each alphabetic character is represented by one byte. Most Western languages, such as English, can be represented by using a single-byte character set.
single-level tracking	Finding all immediate parents where a specific lot has been used (consumed).
single-voyage (spot) charter	An agreement for a single voyage between two ports. The payment is made on the basis of tons of product delivered. The owner of the vessel is responsible for all expenses.
slimer	A script that changes data in a table directly without going through a regular database interface.
smart field	A data dictionary item with an attached business function for use in the Report Design Aid application.
SOC	The Italian term for a Swiss payment format that is accepted by Postfinance.
soft commitment	The number of items that is reserved for sales orders or work orders in the primary units of measure.
soft error	An error from which an operating system or program is able to recover.
software action request (SAR)	An entry in the AS/400 database that is used for requesting modifications to PeopleSoft software.
SOG	The French term for a Swiss payment format that is accepted by Postfinance.
source directory	The path code to the business function source files belonging to the shared library that is created on the enterprise server.
special period/year	The date that determines the source balances for an allocation.

specification merge	The Specification merge is comprised of three merges: Object Librarian merge (via the Object Management Workbench). Versions List merge. Central Objects merge. The merges blend customer modifications with data that accompanies a new release.
specification table merge workbench	During the Installation Workbench process, Specification Table Merge Workbench runs the batch applications that update the specification tables.
specifications	A complete description of an EnterpriseOne object. Each object has its own specification, or name, which is used to build applications.
spot charter	See single-voyage charter.
spot rates	An exchange rate that is entered at the transaction level. Spot rates are not used on transactions between two EMU member currencies because exchange rates are irrevocably fixed to the euro.
stamp tax	In Japan, a tax that is imposed on drafts payable, receipts over 30000 Japanese yen, and all contracts. The party that issues any of the above documents is responsible for this tax.
standalone	Operating or capable of operating independently of certain other components of a computer system.
standard cost	The expected, or target cost of an item, operation, or process. Standard costs represent only one cost method in the Product Costing system. You can also calculate, for example, future costs or current costs. However, the Manufacturing Accounting system uses only standard frozen costs.
standard costing	A costing method that uses cost units that are determined before production. For management control purposes, the system compares standard costs to actual costs and computes variances.
subprocess	A process that is triggered by and is part of a larger process, and that generally consists of activities.
subscriber table	The Subscriber table (F98DRSUB), which is stored on the Publisher Server with the Data Replication Publisher table (F98DRPUB), that identifies all of the subscriber machines for each published table.
summary	The presentation of data or information in a cumulative or totaled manner in which most of the details have been removed. Many systems offer forms and reports that summarize information which is stored in certain tables. Contrast with detail.
super backflush	To create backflush transactions for material, labor, or both, against a work order at predefined pay points in the routing. By doing so, you can relieve inventory and account for labor amounts at strategic points throughout the manufacturing process.
supersession	Specification that a new product is replacing an active product on a specified effective date.

supplemental data	Additional types of data for customers and suppliers. You can enter supplemental data for information such as notes, comments, plans, or other information that you want in a customer or supplier record. The system maintains this data in generic databases, separate from the standard master tables (Customer Master, Supplier Master, and Address Book Master).
supplying location	The location from which inventory is transferred once quantities of the item on the production line have been depleted. In kanban processing, the supplying location is the inventory location from which materials are transferred to the consuming location when the containers are replenished.
system code	A numeric or alphanumeric designation that identifies a specific system in EnterpriseOne software.
system function	[In EnterpriseOne] A named set of pre-packaged, re-usable instructions that can be called from event rules.
table access management (TAM)	The EnterpriseOne component that handles the storage and retrieval of user defined data. TAM stores information such as data dictionary definitions; application and report specifications; event rules; table definitions; business function input parameters and library information; and data structure definitions for running applications, reports, and business functions.
table conversion workbench	During the Installation Workbench process, Table Conversion Workbench runs the table conversions that change the technical and application tables to the format for the new release of EnterpriseOne. It also updates the Table Conversions and Controls detail records to reflect completion.
table design aid (TDA)	An EnterpriseOne GUI tool for creating, modifying, copying, and printing database tables.
table event rules	Use table event rules to attach database triggers (or programs) that automatically run whenever an action occurs against the table. An action against a table is referred to as an event. When you create an EnterpriseOne database trigger, you must first determine which event will activate the trigger. Then, use Event Rules Design to create the trigger. Although EnterpriseOne allows event rules to be attached to application events, this functionality is application-specific. Table event rules provide embedded logic at the table level.
table handle	A pointer into a table that indicates a particular row.
table space	[In relational database management systems] An abstract collection of containers in which database objects are stored.
task	[In Solution Explorer and EnterpriseOne Menu] A user defined object that can initiate an activity, process, or procedure.
task view	A group of tasks in Solution Explorer or EnterpriseOne Menu that are arranged in a tree structure.
termo de abertura	In Brazil, opening terms for the transaction journal.
termo de encerramento	In Brazil, closing terms for the transaction journal.
three-tier processing	The task of entering, reviewing, approving, and posting batches of transactions.

three-way voucher match	The process of comparing receipt information to supplier's invoices to create vouchers. In a three-way match, you use the receipt records, the purchase order, and the invoice to create vouchers.
threshold percentage	In Capital Asset Management, the percentage of a service interval that you define as the trigger for maintenance to be scheduled. For example, you might set up a service type to be scheduled every 100 hours with a threshold percentage of 90 percent. When the equipment accumulates 90 hours, the system schedules the maintenance.
throughput agreement	A service agreement in which a business partner agrees to store and manage product for another business partner for a specified time period. The second partner actually owns the stock that is stored in the first partner's depot, although the first partner monitors the stock level; suggests replenishments; and unloads, stores, and delivers product to the partner or its customers. The first partner charges a fee for storing and managing the product.
throughput reconciliation	Reconcile confirmed sales figures in a given period with the measured throughput, based on the meter readings. This process is designed to catch discrepancies that are due to transactions not being entered, theft, faulty meters, or some combination of these factors. This reconciliation is the first stage. See also operational reconciliation.
token	[In Object Management Workbench] A flag that is associated with each object which indicates whether you can check out the object.
tolerance range	The amount by which the taxes that you enter manually can vary from the tax that is calculated by the system.
TP monitor	Transaction Processing monitor. A monitor that controls data transfer between local and remote terminals and the applications that originated them. TP monitors also protect data integrity in the distributed environment and can include programs that validate data and format terminal screens.
tracing	The act of researching a lot by going backward, to discover its origin.
tracking	The act of researching a lot by going forward, to discover where it is used.
transaction set	An electronic business transaction (EDI Standard document) composed of segments.
transclude	To include the external data in the displayed content through a linking mechanism.
transfer order	An order that is used to ship inventory between branch/plants within your company and to maintain an accurate on-hand inventory amount. An interbranch transfer order creates a purchase order for the shipping location and a sales order for the receiving location.
translation adjustment account	An optional G/L account used in currency balance restatement to record the total adjustments at a company level.
translator software	The software that converts data from an application table format to an EDI Standard Format, and from EDI Standard Format to application table format. The data is exchanged in an EDI Standard, such as ANSI ASC X12, EDIFACT, UCS, or WINS.

tree structure	A type of graphical user interface that displays objects in a hierarchy.
trigger	Allows you to attach default processing to a data item in the data dictionary. When that data item is used on an application or report, the trigger is invoked by an event which is associated with the data item. EnterpriseOne also has three visual assist triggers: Calculator. Calendar. Search form.
two-way voucher match	The process of comparing purchase order detail lines to the suppliers' invoices to create vouchers. You do not record receipt information.
universal batch engine (UBE)	[In EnterpriseOne] A type of application that runs a noninteractive process.
unnormalized	Data that is a random collection of data elements with repeating record groups scattered throughout. Also see Normalized.
user overrides merge	The User Overrides merge adds new user override records into a customer's user override table.
user-defined code (UDC)	A value that a user has assigned as being a valid entry for a given or specific field.
utility	A small program that provides an addition to the capabilities which are provided by an operating system.
variable numerator allocations	A procedure that allocates or distributes expenses, budgets, adjustments, and so on, among business units, based on a variable.
variable quantity	A term that indicates the bill of material relationship between a parent item and its components or ingredients. When a bill of material component has a variable quantity relationship to its parent, the amount of the component changes when the software calculates parts list requirements for different work order quantities. Contrast with fixed quantity.
variance	1. In Product Costing and Manufacturing Accounting, the difference between the frozen standard cost, the current cost, the planned cost, and the actual cost. For example, the difference between the frozen standard cost and the current cost is an engineering variance. Frozen standard costs come from the Cost Components table, and the current costs are calculated by using the current bill of material, routing, and overhead rates. 2. In Capital Asset Management, the difference between revenue that is generated by a piece of equipment and costs that are incurred by the equipment.
versions list merge	The Versions List merge preserves any non-XJDE and non-ZJDE version specifications for objects that are valid in the new release as well as their processing options data.
VESR	Verfahren Einzahlungsschein mit Referenznummer. The processing of an ESR pay slip with reference line through accounts receivable and accounts payable.
visual assist	Forms that can be invoked from a control to assist the user in determining what data belongs in the control.

voucher logging	The process of entering vouchers without distributing amounts to specific G/L accounts. The system initially distributes the total amount of each voucher to a G/L suspense account, where it is held until you redistribute it to the correct G/L account.
wareki date format	In Japan, a calendar format, such as Showa or Heisei. When a new emperor begins to reign, the government chooses the title of the date format and the year starts over at one. For instance, January 1, 1998, is equal to Heisei 10, January 1st.
wash down	A minor cleanup between similar product runs. Sometimes used in reference to the sanitation process of a food plant.
wchar_t	An internal type of a wide character. Used for writing portable programs for international markets.
web server	A server that sends information as requested by a browser and uses the TCP/IP set of protocols.
work order life cycle	In Capital Asset Management, the sequence of events through which a work order must pass to accurately communicate the progress of the maintenance tasks that it represents.
workfile	A system-generated file that is used for temporary data processing.
workflow	According to the Workflow Management Coalition, workflow means “the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.”
workgroup server	A network server usually containing subsets of data that are replicated from a master network server.
WorldSoftware architecture	The broad spectrum of application design and programming technology that PeopleSoft uses to achieve uniformity, consistency, and complete integration throughout its software.
write payment	A step in processing payments. Writing payments includes printing checks, drafts, and creating a bank tape table.
write-off	A method for getting rid of inconsequential differences between amounts. For example, you can apply a receipt to an invoice and write off the difference. You can write off both overpayments and underpayments.
Z file	For store and forward (network disconnected) user, EnterpriseOne store-and-forward applications perform edits on static data and other critical information that must be valid to process an order. After the initial edits are complete, EnterpriseOne stores the transactions in work tables on the workstation. These work table are called Z files. When a network connection is established, Z files are uploaded to the enterprise server; and the transactions are edited again by a master business function. The master business function then updates the records in your transaction files.

z-process	A process that converts inbound data from an external system into an EnterpriseOne software table or converts outbound data into an interface table for an external system to access.
zusammenfassende melding	In Germany, the term for the EU Sales Listing.

Index

A

Allocating memory
 example of, 34
 jdeAlloc, 33

API

 example of, 59, 60
 flat file, 60
 MATHNUMERIC, 58
 third party, 59
 Unicode, 58, 59, 60

B

Business function data structure
 naming standard, 9
Business function programming standards
 introduction, 4

C

Calling internal business functions, 29
Change logs
 updating, 10
Character set
 Unicode, 54
Coding guidelines
 general, 28
Comments
 `/*comment */` style, 10
 alignment, 10
 example of, 10
 readability, 10
Comparison tests
 use of, 35
Compound statements
 alignment of, 12
 declaring variables, 12
 defined, 12
 example of, 13, 14, 15
 number allowed per line, 12
 readability, 12
 use of braces, 12
 use of logical expressions, 12
 use of parenthesis, 12
Compound statements code
 example of, 12
Copying strings
 same or different lengths, 36

Creating

 business function definition, 19
 business function prototype, 18
 C++ comments, 43
 internal function definition, 20
 internal function prototype, 19

D

Data dictionary trigger, 53
 example of, 53
Data structure
 DSDE0022, 52
Data structures
 declaring and initializing, 22
 example of, 23
Data type
 JDEDATE, 46
 MATH_NUMERIC, 45
Declaring
 overview, 16
Declaring and initializing
 data structures, 22
 define statements, 16
 example of, 16, 17, 18, 19, 20, 21, 23, 25
 flag variables, 24
 function prototypes, 18
 input and output parameters, 24
 overview, 16
 standard variables, 23
 typedef statements, 17
 variables, 20
Define statements
 declaring and initializing, 16
 example of, 16, 17

E

Entry point
 defining in main body, 64
 source preprocessing definitions, 64
errors
 data structure, 52
Errors
 DSDE0022, 52
 lpDS, 49
 standard, 50
 text substitution, 51
External business function

- calling, 28
- example of, 28
- External business functions
 - associating, 64
 - header inclusions section, 63

F

- Fetch variables, 24
 - declaring and initializing, 24
- Flag variables
 - declaring and initializing, 24
- Function
 - naming standard, 6
- Function blocks
 - used in compound statements, 12
- Function calls
 - data types, 28
 - external, 28
 - internal, 29
 - jdeCallObject, 28
 - return value, 28
 - use of space, 28
 - with long parameter lists, 28
- Function clean up area
 - example of, 36
 - to release memory, 36
- Function exit points
 - example of, 37
 - number of, 37
 - use of, 37
- Function prototype
 - example of, 18, 19, 20
- Function prototypes
 - declaring and initializing, 18
 - placement of, 18
 - variable names
 - data types, 18

G

- GENLNG
 - to receive an address, 32
 - use of, 32

H

- header file
 - change log, 10
 - naming standard, 6
- Header file
 - standard, 61
 - template, 61
- Hungarian notation for variables

- example of, 8

I

- Indentation
 - of code, 11
 - when to indent, 11
- Indenting code
 - example of, 11
 - readability, 11
- Initializing variables
 - example of, 21
- Initializing overview, 16
- Input and output parameters
 - declaring and initializing, 24
- Input parameters
 - for error messages, 24

J

- JDB Errors, 48
- JDE Cache Errors, 48
- jdeapp.h
 - used in defines and typedefs, 17
- jdeCallObject
 - calling business functions, 28, 29
 - to map data structure errors, 52
- JDEDATE
 - data type, 46
- jdeMemset, 55
 - example of, 56

L

- Logical expressions
 - in compound statements, 12

M

- MATH_Numeric
 - data type, 45
- MATH_NUMERIC
 - used in variable declarations, 21
 - when assigning variables, 46
- MathCopy
 - when assigning MATH_NUMERIC variables, 46
- Memcpy
 - when assigning JDEDATE variables, 47
- Memory
 - use of jdeAlloc, 33
- Memory function
 - example of, 56
 - unicode, 55

- Memset data structure to NULL
 - example of, 23
- memset function, 55
- Multiple logical expressions
 - example of, 15

N

- Naming standard
 - business function data structures, 9
 - defines
 - typedefs, 17
 - examples of, 8
 - flag variables, 24
 - functions, 6
 - introduction, 6
 - source and header files, 6
 - standard variables, 23
 - variables, 7

O

- Offsets
 - Unicode, 58

P

- Parenthesis
 - used in compound statements, 12
- Pointer
 - example of, 57
 - Unicode, 56
- Program flow
 - overview, 5

R

- Readability
 - comments, 10
 - compound statements, 12
 - example of, 10, 11, 12, 13, 14, 15
 - indenting code, 11
 - overview, 10
 - source and header change logs, 10
- Releasing memory
 - example of, 34
- Removing an address
 - example of, 33
- Retrieving an address
 - example of, 33

S

- Source
 - template, 65

- source file
 - change log, 10
 - naming standard, 6
- Source file
 - standard, 61
- Source preprocessor section
 - example of, 64
- Standard variables
 - boolean flag, 23
 - declaring and initializing, 23
 - example of, 25
 - flag variables
 - example of, 23
- StartFormDynamic
 - example of, 16
- Storing an address
 - example of, 32
- strcpy vs. strncpy
 - when to use, 36
- String function
 - example of, 55
- String functions
 - Unicode, 55
- Syntax
 - Unicode, 54

T

- Template
 - standard header, 61
 - standard source, 65
- Typecasting
 - in prototypes, 34
 - use of, 34
- Typedef statements
 - declaring and initializing, 17
 - example of, 17

U

- Unicode
 - API, 58, 59, 60
 - character set, 54
 - standards, 54
 - syntax, 54
- User-defined data structure
 - example of, 17
- Using braces
 - example of, 13, 14
 - for ease in subsequent modifications
 - example of, 14
 - to clarify flow
 - example of, 14

- Using standard variables
 - example of, 25
- Using StartFormDynamic
 - example of, 16

V

- Variable
 - naming standard, 7
- Variable declarations
 - description, 21
 - initial value, 21
 - initialization of, 21
 - memset data structure to NULL, 21

- number per line
 - alignment, 21
 - placement in function, 20
 - use of MATH_NUMERIC variables, 21
 - use of NULL pointers, 21
- Variable initialization
 - example of, 21
 - types, 21
- Variable names
 - Hungarian notation, 8
- Variables
 - declaring and initializing, 20

