

PeopleSoft®

EnterpriseOne 8.93
Development Tools
PeopleBook

May 2004

EnterpriseOne 8.93
Development Tools
SKU TT893DT0504

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
PeopleSoft EnterpriseOne Development Tools	4
EnterpriseOne Acronyms	4
Objects and Applications.....	5
Objects	5
Applications	6
How to Build an Application	6
The Development Cycle.....	7
How EnterpriseOne Stores Objects	8
The EnterpriseOne Tool Set	9
Fundamentals	13
Data Items in the Data Dictionary	13
How the Data Dictionary is Used at Runtime.....	14
Naming Data Items	15
Storing the Data Dictionary and Data Dictionary Items	15
Glossary Items	16
Error Messages.....	16
Understanding Default Triggers	16
Using the Data Dictionary	17
Defining a Data Item	17
Table Design	31
Adding a Table	31
Table Naming Conventions.....	32
Working with Table Design	33
Working with Tables.....	35
Viewing the Data in Tables	37
Business View Design.....	40
Table Join.....	40
Table Union	41
Select Distinct.....	42
Indices	42
Adding a Business View.....	42
Working with Business View Design	44
Data Structures	51
System-Generated Data Structures	51
User-Generated Data Structures	52

Working with Interconnection Data Structures	52
Creating a Data Structure	54
Defining a Data Structure	57
Cross Reference Facility	59
Event Rules	65
Event Rules Design	65
Prerequisites	65
Controls	65
Events	66
Form Processing	66
Event Rules	66
Event Rule Buttons	67
Event Rules Based on Form Type	68
Runtime Processing	68
Working with Event Rules Design	91
Working with If and While Statements	96
Working with Event Rule Variables	100
Creating Form Interconnections	119
Creating Report Interconnections	121
Creating Assignments	122
Table I/O	127
Table Event Rules	135
Creating Dynamic Overrides	137
Working with Asynchronous Processing	138
BrowsER	144
Using Visual ER Compare	146
Business Functions	149
Understanding Business Functions	149
Components of a Business Function	150
How Distributed Business Functions Work	152
Creating Business Function Event Rules	154
C Business Functions	157
Understanding Header File Sections	158
Understanding the Structure of a Business Function Source File	163
Using Application Programming Interfaces (APIs)	168
Working with Business Functions	176
Creating and Specifying Custom DLLs	178
Working with Business Function Builder	179
Transaction Master Business Functions	190
Master File Master Business Functions	205
Business Function Documentation	208
Caching	216
JDECACHE	216
When to Use JDECACHE	216
Performance Considerations	217
The JDECACHE API Set	217
Working with JDECACHE	219
Calling JDECACHE APIs	219
Setting Up Indices	220
Initializing the Cache	223

Using an Index to Access the Cache	224
Using the jdeCacheInit/jdeCacheTerminate Rule	225
Using the Same Cache in Multiple Business Functions or Forms	225
JDECACHE Cursors	226
Opening a JDECACHE Cursor	226
HJDECURSOR	227
JDECACHE Dataset	227
Updating Records	229
Deleting Records.....	230
The jdeCacheFetchPosition API	230
The jdeCacheFetchPositionByRef API	230
Resetting the Cursor	231
Closing a Cursor	231
JDECACHE Multiple Cursor Support.....	231
JDECACHE Partial Keys.....	232
How a JDECACHE Partial Key Works	233
JDECACHE Standards	233
Cache Business Function Source Description	233
Cache Programming Standards.....	234
Additional Features	238
Processing Options	238
Processing Options Templates	238
Defining a Processing Options Data Structure (Template)	240
Attaching a Processing Options Template	243
Transaction Processing.....	244
Commits and Rollbacks	245
Transaction Processing.....	245
Working with Transaction Processing	249
Setting the jde.ini for Lock Manager	252
Record Locking	253
Understanding Record Locking.....	254
Currency.....	255
Currency Implementation.....	255
Advantages	256
Working with Currency	256
Tips of the Day	260
Working with Tips of the Day	260
Messaging	262
Message Types.....	262
Error Messages.....	263
Workflow Messages	263
Level Messages	263
Information Messages.....	263
Understanding Error Handling.....	264
Event-Driven Model.....	264
Error Setting	265
Resetting Errors	267
Multilevel Error Messaging for C Business Functions.....	268

Working with Error Messages	270
Locating an Existing Error Message	270
Creating a Simple Error Message	271
Creating a Text Substitution Error Message	271
Attaching an Interactive Message Data Item	273
Working with the Send Message System Function	273
Defining an Active Message	275
Batch Error Messages	276
Batch Error Messaging	276
Level-Break Messages	276
Understanding How Level-Break Messages Work	276
Creating a Level-Break Message	279
Creating a Data Dictionary Item for a Level-Break Message	279
Creating a Data Structure for the Data Dictionary Item	280
Creating a Data Structure for a Level-Break Message Business Function	281
Creating a Level-Break Business Function	282
Calling the Work Center Initialization API	287
Calling the Processing Work Center APIs	288
Terminating the Work Center Process	290
Debugging	291
Overview of the Debugging Process	291
Troubleshooting: Development Tools	293
Interpretive and Compiled Code	296
Working with the Event Rules Debugger	297
Understanding the Event Rules Debugger	297
Debugging an Application	300
Setting Breakpoints	301
Debugging Business Functions By Using Microsoft Visual C++	301
Debugging Business Functions Attached to Interactive Applications	301
Debugging Business Functions Attached to Batch Applications	302
Working with the Visual C++ Debugger	303
Useful Features of the Visual C++ Debugger	303
Example: Debugging with the Visual C++ Debugger	305
Customizing the Environment	305
Debugging Strategies	306
Is the Program Ending Unexpectedly?	306
Is the Application Encountering Errors?	306
Is the Output of the Program Incorrect?	307
Where Else Could the Problem Be Coming From?	307
Is the Function Being Called as Expected?	307
Debug Logs	307
SQL Log Tracing	308
Debug Tracing	309
System Function Tracing	309
Developing Web Applications	311
The HTML Client	313

Performance	315
Triggers	315
Overrides	315
Validation	315
Table Design Performance	315
Index Considerations	316
Index Limitations for Various Databases	316
Access32	316
SQLSERVER	317
DB2 for OS/400	317
Oracle	317
Key Column Violation	318
Specification File Corruption	318
Table I/O Objects	319
Business View Performance	319
Should I Use a Single-Table or Multiple-Table Business View?	319
Should I Restrict the Number of Fields in a Business View?	320
Unions	320
Data Structure Performance	320
Data Selection and Sequencing	321
Form Design	321
Find/Browse	321
Header Detail and Headerless Detail	322
Batch Application Performance	322
Event Rules Performance	322
Using jdeCache Instead of Work Tables	323
Using JDBFetchNext versus Multiple JDBFetch Statements	323
Business Function Performance	323
Error Messaging Performance	325
Transaction Processing Performance	325
PeopleSoft EnterpriseOne Modification Rules	326
What an Upgrade Preserves and Replaces	327
General Rules for Modification	327
Interactive Applications	327
Reports	330
Application Text Changes	331
Table Specifications	331
Control Tables	332
Business Views	332
Data Structures	333
Event Rules	333
Versions	335
Business Functions	335

Date Reference Scan	336
Business Function Date Reference Scan	336
Event Rule Date Reference Scan	337
EnterpriseOne PeopleBooks Glossary	338
Index	371

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.

PeopleSoft EnterpriseOne Development Tools

The PeopleSoft EnterpriseOne Development Tools are an integrated set of application development tools. These tools allow business analysts to develop complete interactive and batch applications, such as forms and reports. The tools simplify the development process and limit the amount of programming necessary to create applications.

EnterpriseOne Development Tools also allow you to use the stability of EnterpriseOne methodology and the ease of the Microsoft Windows interface to create applications for client/server environments.

EnterpriseOne Acronyms

Many of the acronyms that are commonly used in EnterpriseOne are described in the following table:

APPL	Application
BDA	Business View Design Aid
BSFN	Business Function
BSVW	Business View
CRP	Conference Room Pilot
DD	Data Dictionary
DLL	Dynamic Link Library
DS or DSTR	Data Structure
ER	Event Rule
FDA	Form Design Aid
NER	Named Event Rule
OCM	Object Configuration Manager
OL	Object Librarian
OMW	Object Management Workbench
QBE	Query by Example
RDA	Report Design Aid
SAR	Software Action Request

Specs	Specifications
TAM	Table Access Management
TBLE	Table
TC	Table Conversion
TDA	Table Design Aid
TER	Table Event Rule
UBE	Universal Batch Engine
UTB	Universal Table Browser
WF	Workflow

Objects and Applications

You use EnterpriseOne tools to build objects. You use objects to build applications. The following topics describe objects and applications in detail.

Objects

By industry standards, an object is a self-sufficient entity that contains data as well as the structures and functions that are used to manipulate that data. In EnterpriseOne, an object is a reusable entity that is based on software specifications created by the EnterpriseOne Tools.

A specification is a complete description of an EnterpriseOne object. Each object has its own specification, which is stored on both the server and the workstation. Some specifications describe types of objects; for example, the data structure specification describes a business function data structure, a processing option structure, and a media object structure.

EnterpriseOne architecture is object-based, which means that discrete software objects are the basis for all applications and that developers can reuse the objects in multiple applications. This use of objects (applications being broken down into smaller components) allows PeopleSoft to provide true distributed processing. Developers create objects using EnterpriseOne Tools.

Examples of EnterpriseOne objects include the following:

- Batch applications
- Business functions (encapsulated routines)
- Business views
- Data dictionary items
- Data structures
- Event rules

- Interactive applications
- Media objects
- Tables

Applications

An application is a collection of objects that performs a specific task. You use EnterpriseOne Tools to build the following standard groups of related applications:

- Architecture, engineering, and construction
- Distribution
- Energy and chemical systems
- Financial
- Workforce management
- Manufacturing
- Technical

Applications share a common user interface because they are all generated through EnterpriseOne Tools. Applications refer to both interactive and batch applications. All of the following are applications:

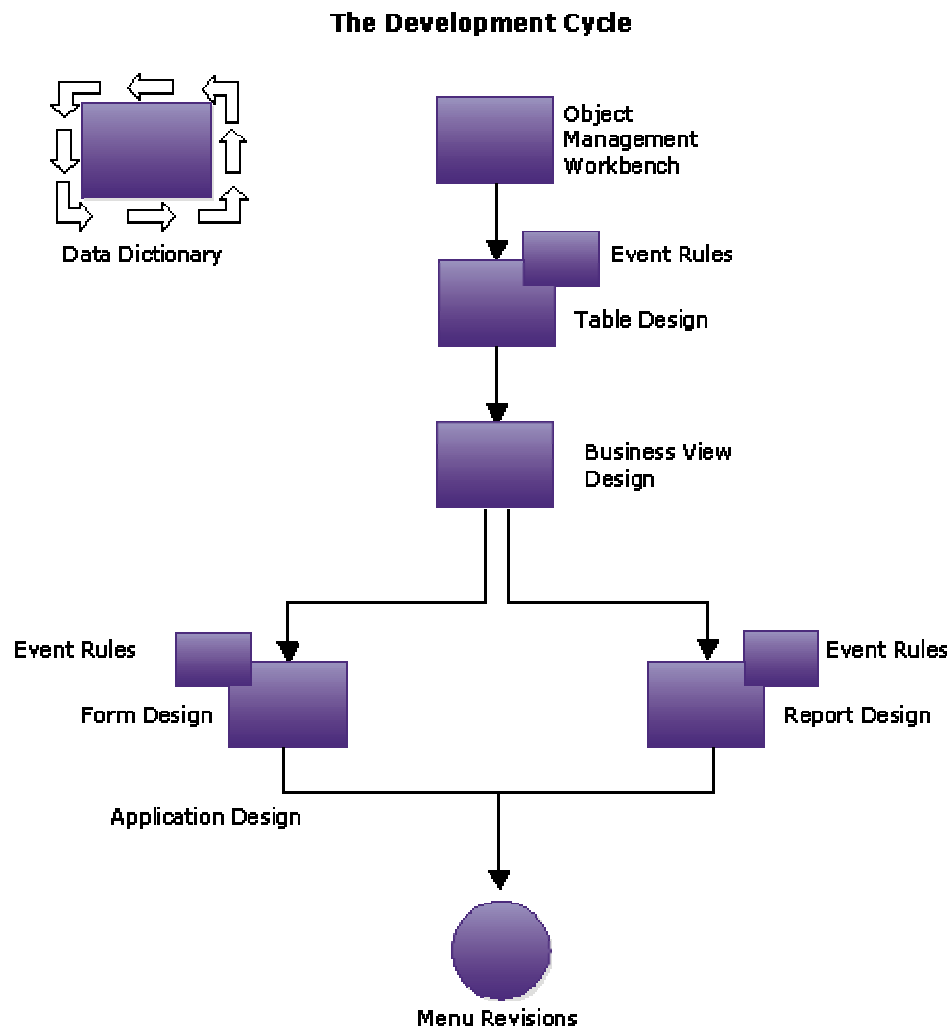
- Address Book Revisions
- Sales Order Entry
- General Ledger Post
- Trial Balance Report

How to Build an Application

You use the EnterpriseOne Tools to build your applications. You always begin your application development from the Object Management Workbench. You might not use every tool to create an application. For example, if you don't need to add or modify data items you would proceed to Table Design from the Object Management Workbench. If one or more existing database tables contain all of the data items that you want to include in your application, you can skip the step of designing a table and proceed to Business View Design.

The Development Cycle

The following illustration of the software development cycle shows the relationships among various tools.



How EnterpriseOne Stores Objects

EnterpriseOne stores objects in the following two places:

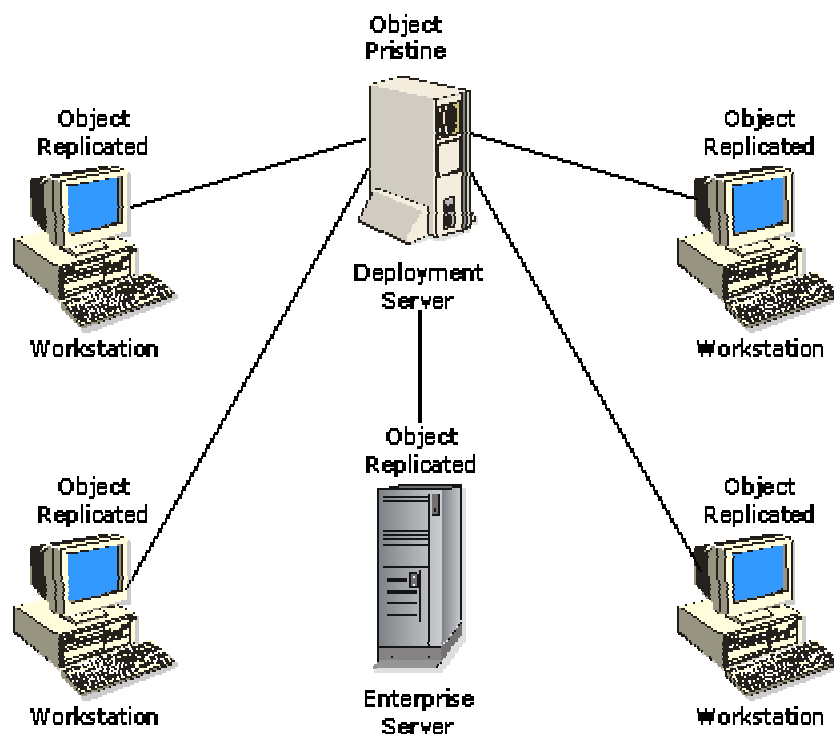
- A central-storage server stores central objects

Central objects reside in a central location from which you can deploy them. Other objects, such as specifications, are stored in a relational database. Still others, such as DLLs and source code, are stored on a file server.

- Any machine (workstation or server) that runs EnterpriseOne stores replicated objects

A copy (replicated) set of the central objects must reside on each workstation and server that runs EnterpriseOne. The path code indicates the directory in which these objects are located.

How PeopleSoft EnterpriseOne Stores Object



To move objects between the server and workstation, you use the design tool for the specific object type and you use check-in and check-out. When you create an object, it initially resides on your workstation. Unless you check it into the server, it is available only to you. After you check it into the server, it is available for other users to check out.

When you check out an object, all object specification records (a collection of data that defines an EnterpriseOne object) are replicated from the server to your workstation.

The EnterpriseOne Tool Set

EnterpriseOne Tools is a set of several tools that you use to create an interactive or batch application. These tools are detailed in the following topics.

Object Management Workbench

The Object Management Workbench manages all objects. Developers use the Object Management Workbench to check out objects from a central development environment and to copy the objects to their desktops. They can then use the tools to revise objects and check in the objects so that others can access the objects. Developers access PeopleSoft EnterpriseOne Tools through the Object Management Workbench.

Data Dictionary

Just as a dictionary contains word definitions, the PeopleSoft EnterpriseOne data dictionary is a central repository that contains data item definitions and attributes. Attributes determine the way in which a data item does the following:

- Appears on reports and forms (factors such as the number of decimal positions and whether default values appear)
- Validates data entry within an application
- Assigns column and row descriptions
- Provides text for field-sensitive help
- Is stored in a table

The data dictionary is active because the changes that you enter in it are automatically reflected in applications. You do not need to recompile the software to see the changes.

Table Design

A relational database table stores data that an application uses. You can create one or more tables for use in an application. To create a table, you select data items for the table and then assign key fields as indices for retrieving and updating data.

Table Design Aid creates an .H file. You use OMW to check this file into source.

Business View Design

Business views are the link between applications and data. A business view defines the data items from one or more tables that an application uses. After you create the tables, you select the data items that you want to include in your business view from one or more tables. For business views, you typically select only the columns that you need in the application, which increases performance by reducing the amount of data that must move over a network. For example, from a table that contains all employee data, you can create a business view that contains only employee names and addresses.

Form Design

An interactive application allows a user to access a database (based on a business view). Users typically use a form to access an interactive application in which they add, modify, or view data.

To create an application, you determine the type of forms that your application requires, and associate each form with a business view. To design forms, you add controls such as a grid, edit fields, push buttons, and radio buttons.

Several standard form types have much of the required processing already established. Examples include:

Find/Browse	Used for inquiry forms, such as Work with Sales Orders.
Fix/Inspect	Used to add or modify a single record. Sales Order Header is an example of a fix/inspect form.
Header Detail	Used to modify multiple records at a time (particularly on normalized tables). Sales Order Detail is an example of a header detail form.
Headerless Detail	Used to modify multiple records in a table that is not normalized. Voucher Entry is an example of a headerless detail form.
Search and Select	Used to automatically retrieve data in a visual assist field.
Parent/Child	Used to display multiple records that have a parent/child relationship in an Explorer-like tree view. Bill of Material is an example of a parent/child form.
Message	Used to display messages or request action. Delete Confirmation is an example of a message form.

Report Design

You use Report Design to create reports and batch processes. In Report Design, you design sections instead of forms. To create a report, you determine the data that you want to appear on the report. You then attach event rules that provide business logic and specify processing options that control the format, page breaks, report totaling, and how the report processes data. Examples of reports and batch processes are:

- Sales Update
- General Ledger Post
- Trial Balance by Cost Center
- Invoices
- Table Conversions
- Financial Reporting

Business Function Design

A business function is an encapsulated, reusable routine that can be called through event rules. Code for business functions can be written in most industry-standard third-generation languages. You use Business Function Design to write business functions that provide background processing for the handling of specialized tasks that an application needs, such as specialized editing for a field.

Event Rules

Event rules are logic statements. You create event rules and attach them to events. Events are activities that occur in an application, such as entering a form, exiting a field, exiting a row, or initiating a page break. Event rules respond when the user or the system initiates an event. Events are attached to controls, such as a particular field, the entire form, the grid, or a report section.

You use event rules to create complex business logic without the difficult syntax that comes with most programming languages. Examples of business logic that you can accomplish with event rules include:

- Perform a mathematical calculation
- Perform table I/O (fetch, insert, delete)
- Pass data from a source field on a form to a target field on another form
- Connect two forms or applications
- Hide or display a control by using a system function
- Evaluate if/while and else conditions
- Assign a value to an expression in a field
- Create variables (work fields) as you are working
- Perform a batch process after completing an interactive application
- Attach a business function or system function
- Initiate workflow processes

The two types of event rules are:

- Business function event rules
- Embedded event rules

Business Function Event Rules

Business function event rules are encapsulated, reusable business logic created through event rules rather than C programming. Business function event rules are stored as objects and are compiled.

Embedded Event Rules

Embedded event rules are specific to a particular table, interactive application, or batch application. They are not reusable. Examples include creating form-to-form calls, hiding a field based on a processing option value, and calling a business function. You can place embedded event rules in application event rules (interactive or batch) or in table event rules. The distinctions are explained in the following table:

Application event rules (interactive or batch)

Allow you to add business logic that is specific to a particular application. Interactive applications connect event rules using the Form Design Aid; batch event rules use Report Design.

Table event rules

Allow you to create EnterpriseOne database triggers, or rules that are attached to a table and are executed through Table Design Event Rules. The logic attached to a table is executed whenever any application initiates that database action.

For example, to maintain referential integrity, you might have rules in a master table that delete all children when a parent is deleted. Any application that initiates a delete of that table does not need to include the parent/child logic because that logic exists at the table level.

Processing Options

Processing options control how interactive and batch applications process data. You can have several versions of a set of processing options. Typically, each version differs from the other versions.

For example, you can use processing options to do the following:

- Determine the sequence of the forms in an application
- Set default values
- Enable and disable special processing, such as currency or kit processing in Sales Order Entry

Fundamentals

This topic discusses the basic concepts and tools that you need in order to start developing applications. It includes the preliminary steps that you must take before you design an application.

Data Items in the Data Dictionary

A data item identifies a unit of information. The data item definition defines how the item can be used and includes information such as the type of item and its length.

Because the data dictionary is dynamic, any changes that you make to a data item are effective immediately for all applications that include the data item. Applications access the data dictionary at runtime and immediately reflect modifications to data item attributes such as field descriptions, column headings, decimals, and edit rules.

You use the data dictionary to create, view, and update attributes for data items. You can copy a data item with similar attributes and modify it for your specific needs. This method can be quicker and easier than creating a new data item, but if you use this method you must distinguish between the original and the copy. You distinguish between them by modifying the alias.

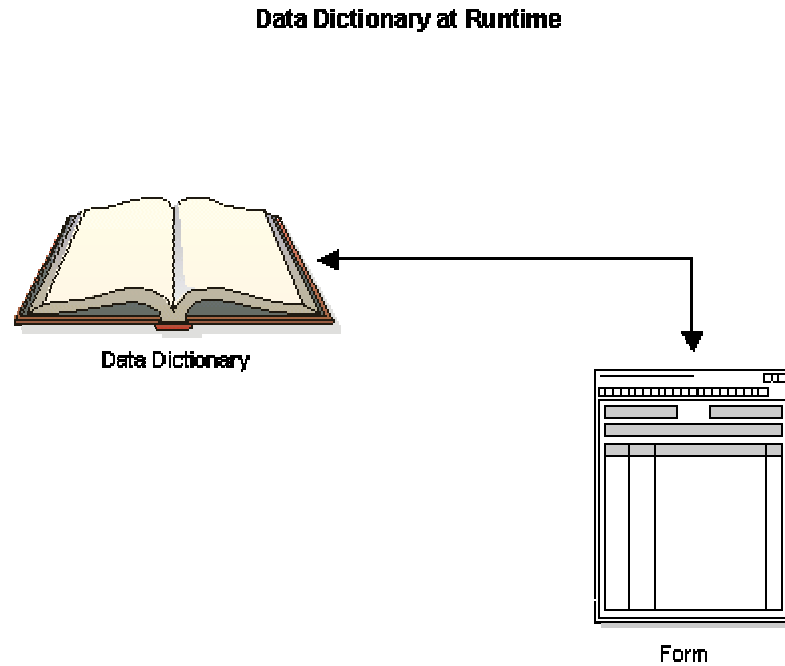
Because changes to a data item are immediately reflected throughout the PeopleSoft EnterpriseOne tools at runtime, remember that changing the type and attributes of a data item might affect how your data is stored and cause discrepancies among records.

Caution

The data dictionary does not verify whether a data item is used by an application when you delete it. If you delete a data item that an application uses, that application will fail.

How the Data Dictionary is Used at Runtime

The following graphic illustrates how the system processes certain information about data items:



At runtime, EnterpriseOne applications (such as those in Accounts Payable or Sales Order Management) access the data dictionary and reflect the assigned attributes in any of the following fields:

- Display Decimals
- File Decimals
- Alpha Description
- Data Type
- Size
- Glossary
- Allow Blank Entry *
- Upper Case Only *
- All Triggers *
- Row and Column Headings *

The application retrieves field information from the data dictionary. Fields marked with an asterisk (*) can be overridden in Form Design and Report Design. In these instances, the application retrieves the overrides, if any exist.

Naming Data Items

The following graphic illustrates the naming components for a data item:

Naming Components for a Data Item

Data Item Name		Alias	Alpha Description	
Company	=	CO	=	Company
Cost Center	=	MCU	=	Business Unit

After a data item is created, you can change only the alpha description; you cannot change the data item name and alias. When you add a data item, the data dictionary validates the data item and alias fields to ensure that they are unique.

Storing the Data Dictionary and Data Dictionary Items

Core data dictionary information is stored in the following two places:

- The central object data dictionary is stored in a relational database. All changes to the data dictionary that will be replicated must be done here.
- The replicated data dictionary allows each workstation to have a set of data dictionary tables stored in specification tables on the client machine.

Data dictionary items reside on enterprise (logic) servers in relational database tables. Workstations retrieve from the publisher data dictionary (the relational database tables) only those data items that are necessary for the applications being used. Replication occurs when you use an application for the first time after installing PeopleSoft EnterpriseOne. The data dictionary information is stored on your workstation in a permanent cache under the same local path code and spec directory as the following global tables:

- glbltbl.xdb (references for the data)
- glbltbl.ddb (the data items)

If data items have been changed and you want to replicate the changes to workstations immediately, you must use the Data Replication (P98DREP) application. When you have data replication set up and an item is changed, the next time that a subscriber machine signs on to the system, the permanent cache for that data item will be deleted. Then, the next time that a user accesses an application that has the changed data item, the system detects that the information is not in the permanent cache and retrieves the information from the publisher data dictionary (the relational database tables).

If you use PeopleSoft EnterpriseOne software in coexistence with PeopleSoft World software, you must maintain two data dictionaries. You cannot share one dictionary because, in PeopleSoft World software, the \$ is reserved for business partners and is used for the beginning of an alias. The \$ does

not compile and, thus, cannot be shared with PeopleSoft EnterpriseOne software. Some of the file formats in PeopleSoft World software are also different.

See Also

- ❑ *Data Dictionary Administration* in the *System Administration Guide* for information about administering data dictionary items

Glossary Items

Glossary items are items that cannot be attributes in tables. Glossary items are typically used as information messages.

Error Messages

Error messages used in EnterpriseOne are stored as data items. Error messages do not use all the fields that are required for regular data items, so you need to use the data dictionary glossary item application to display error messages.

Understanding Default Triggers

A default trigger is an editing or display routine that is attached at the dictionary level and initiated at runtime. Default triggers are reusable objects and, therefore, automatically associated with each application that uses the data item. Default triggers save time and increase the usefulness of your code because you can create the business logic only once and then use it within multiple applications. Default triggers ensure accuracy and integrity of data across all applications.

You use triggers to do the following:

- Establish field default values.
- Link data items to a user defined code (UDC) table of valid values.
- Activate a visual assist search program when a user positions the cursor in a field. The types of visual assists are:
 - Calendar
 - Calculator
 - Search for and select another file
 - UDC
- Establish rules and procedures that are embedded in the editing and formatting of the data for a field.
- Determine a next number scheme that developers should use when assigning a number to data.

Using the Data Dictionary

You can create new data dictionary items and view existing ones with the Object Management Workbench or with the Data Dictionary Application program (P92001). After you create a data dictionary item, use the Data Dictionary Application program to define jargon and language translations for it.

► To use the data dictionary

From the Data Dictionary menu (GH951), choose Work With Data Dictionary Items.

On Work With Data Items, complete any of the following fields and click Find.

- Search Description
- Data Item
- Description
- Alias
- Glossary Group
- System Code
- Product Code Reporting

Defining a Data Item

You define a data item, including its specifications, when you add, modify, or copy a database data item.

Creating a Data Item

Data dictionary items are the foundation of PeopleSoft EnterpriseOne objects. You create data dictionary items for use as fields on a form, columns in a table, fields in a business view, members of a data structure, fields on a UBE, and so on.

► To create a data item

1. From the Object Management Workbench, click Add.
2. On Add J.D. Edwards Object to the Project, click the Data Item option in the Control Table Objects portion of the form, and then click OK.

The system prompts you to create a regular data item or a glossary data item. Glossary data items are used primarily for batch error messages.

3. To create a regular data item, click No.
4. On Data Items Specification, complete the steps to name the data item.

Data Dictionary Naming Conventions

You must adhere to data dictionary naming conventions to ensure database integrity and prevent data items from being overwritten by other data items.

Data Item Name

The data item name is a 32-character, alphabetical field that identifies and defines a data item. You must allow enough space in the field name for a 30 percent expansion of the English text when translated.

The data item name forms the C-code data name (for example, AddressNumber) that is used in business functions, data structures, and event rules. You can also identify a data item by its alias or alpha description.

When creating an EnterpriseOne data item, do not use Y or Z as the first character of the data item name; Y and Z are reserved for Partners in Development business partners.

Blanks and the following characters are not allowed in a data item name:

%

&

+

Note

After you add a data item, you cannot change its name.

Data Item Name for an External Data Dictionary Item

When you create an external data item, you must use Y or Z as the first character of the data item name to distinguish an external data dictionary item from a PeopleSoft EnterpriseOne data dictionary item.

The data item name is a maximum of 32 alphanumeric characters and uses the following format:

Yssssssssssssssssssssssssssssssssssssssss

Where:

Y or Z designates an external data dictionary item and is the first digit of any external system code assigned by EnterpriseOne

sss is the system code number

This number is 55xx through 59xx for enterprise-level development of new modules, 60xx through 69xx for PeopleSoft EnterpriseOne custom development, or a number assigned by the Partners in Development system administrator.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx is a unique name for the data item

Data Item Alias

The data item alias is an 8-character alpha code. If the data dictionary item is exclusive to EnterpriseOne applications, the alias is five to eight characters long. When you add a data item that will be used in a table by an RPG program, the data item alias must not exceed 4 characters. You can identify a data item by its data item name or an alpha description.

The data item alias is used in searches, database routines (application program interfaces used in business functions), and Table Design when you create a table. A prefix is added to the alias of each table to make it unique to the table. For example, ABMCU indicates that MCU is within the Address Book Master table (F0101).

When you assign an alias, do not begin the alias with TIP or TERM. Aliases that begin with TIP are reserved for EnterpriseOne tips information; aliases that begin with TERM are reserved for term glossaries that are included in EnterpriseOne documentation.

Blanks and the following characters are not allowed in the data item name:

%

&

+

Note

After you add a data item, you cannot change its name.

Alias for an External Data Dictionary Item

An external data dictionary item is one that is created by a developer outside of PeopleSoft for use within EnterpriseOne. For external data items, the data dictionary alias can contain a maximum of eight alphanumeric characters. External data items use the following format:

Ysssdddd

Where:

Y or Z designates an external data dictionary item and is the first digit of any external system code assigned by EnterpriseOne

sss is the system code number

This number is 55xx through 59xx for enterprise-level development of new modules, 60xx through 69xx for PeopleSoft custom development, or a number assigned by the Partners in Development system administrator.

dddd is a unique name for the data item

Naming a Data Item

After you have created a base data item, you must assign a unique name and attributes to this new data dictionary item before it can be included in an application.

► To name a data item

1. On Data Item Specifications, complete the following fields:
 - Data Item
 - Alias
 - Glossary Group
2. Complete the steps to define general information.

See Also

- *Defining General Information in Development Tools Guide*

Defining General Information

After you name a data item, you must provide additional general information on Data Item Specifications.

► To define general information

1. On Data Item Specifications, click the Item Specifications tab.
2. Complete the following required fields:
 - Description
 - Data Type
 - Product Code
 - Control Type
 - Row Description

This description is for the base language only, unless you update the description for another language.
 - Column Title

This description is for the base language only, unless you update the description for another language.
 - Size
 - File Decimals
 - Display Decimals

3. Complete the following optional fields:

- Class
- Item Occurrences

Item occurrences is used for arrays. It allows you to create an item as a child of another item. The data dictionary verifies that attributes are consistent between the parent and the child. If you change the parent item, the changes are duplicated in the child items. The data item names use the parent data item name and a number, such as a parent item ABC and child items ABC1, ABC2, and so on.

4. Click any of the following options:

- Upper Case Only
- Row Security
- Allow Blank Entry
- Auto Include
- Do Not Total

5. Complete the steps to attach default triggers.

See Also

- ❑ *Attaching Default Triggers* in the *Development Tools Guide*

Attaching Default Triggers

You use triggers to initiate display and edit routines that are associated with data items at application runtime.

► To attach default triggers

1. On Data Item Specifications, click any of the following tabs to attach the applicable triggers to a data item:

- Default Value
- Visual Assist
- Edit Rule
- Display Rule
- Next Number

2. On each tab, click the appropriate option for the type of default value that you want to attach.

After you click certain options, the system displays additional fields into which you supply the appropriate default values.

Although you can override any of these triggers in Form Design in the Edit Control Properties, you should anticipate how they would most often be used.

Attaching a Default Value Trigger

You use the default value trigger for fields in which a default value is appropriate for most situations. If a value other than the default value is passed to this field through a processing option, business function, form interconnection, or data structure, or if the user enters a different value, the default value will not be used. A default value trigger automatically inserts a specified default value in a field when the following conditions are met:

- Control is exited
- Dialog is initialized

► To attach a default value trigger

1. On Data Item Specifications, click the Default Value tab.
2. Click the following option:
 - Default Value
3. Enter the default value.

Attaching a Visual Assist Trigger

The following types of visual assist triggers are available:

- Search form
- Calculator
- Calendar

Use a search form trigger to link a field to a search form for locating valid values. The search form must exist before you attach a search form trigger.

► To attach a visual assist trigger

1. On Data Item Specifications, click the Visual Assist tab.
2. On Visual Assist, click one of the following options:
 - Calculator
 - Calendar
 - Search Form
 - JDE UTime
3. If you click Search Form, click the Browse button to choose the form that you want to appear for valid values.

If you attached a Search Form assist trigger, when a user clicks the Search Form button at runtime, the form that you specify here appears. The user enters search criteria, and the system returns valid values. These Search and Select forms are based on files, not UDC tables.

Attaching an Edit Rule Trigger

You use edit rule triggers to validate field values based on business functions or rules. For example, you can define a rule that does the following:

- Validates and compares a field with a particular value
- Ensures that a field value is within a specified range of values
- Links a field to a specific UDC Search and Select form
- Checks for Y and N values

► To attach an edit rule trigger for a business function

1. On Data Item Specifications, click the Edit Rule tab.
2. On Edit Rule, click one of the following options:
 - Business Function
 - Rule
3. If you clicked the Business Function option, click the Browse button to choose from available business functions.
4. On Business Function Search, locate a business function.
5. To verify the purpose of the function, choose the function and choose Attachments from the Row menu.

The system displays a Media Objects form that contains usage notes for the business function. Close the form after you have reviewed the information.

Note

You must create the business function if it does not exist.

6. On Business Function Search, click Select to choose the business function.
7. If you clicked the Rule option, click the search button to choose from the available rules.

These rules can include user defined codes such as:

EQ	Equal
GE	Greater or Equal
GT	Greater
HNDL	Table Handle
LE	Less Than or Equal
LT	Less Than

NE	Not Equal
NRANGE	Not Between
RANGE	Between
UDC	User Defined Code
VALUE	In a List
ZLENGTH	Allocated Length (VARLEN flds)

Attaching a Display Rule Trigger

You use a display rule trigger to format data. You attach a display rule trigger based on either a business function or a user defined code. The following two types of display rule triggers are available:

- Business Function
- Rule

► To attach a display rule trigger

1. On Data Item Specifications, click the Display Rule tab.
2. On Display Rule, click the following option:
 - Rule
3. Click the Search button (for the field that appears) to locate the appropriate code from available user defined codes.

Choose from the following values:

- *RAB** Right-adjusts the value and precedes it with blanks. Data items that define business units use this rule.
- *RABN** Right-adjusts the value and precedes it with blanks. Data items that do not define business units use this rule.
- *RAZ** Right-adjusts the value and precedes it with zeroes. For example, Company appears as 00001.
- CODE** Uses the specified Edit Codes to format numeric fields. See UDC 98/EC for a list of valid codes. The code should be entered into the parameter field.
- MASK** Embeds the specified characters within the data when it appears. For example, to display Social Security number (SSN_) with embedded dashes, the mask parameter would appear as follows:
bbb-bb-bbbb, where *b* corresponds to each digit in the Social Security number
 Mask can be used only with char or string data item types.

Attaching a User Defined Code Trigger

You attach a user defined code trigger to use a UDC table to validate data for a specific field. If you attach such a trigger to a field, only the values in the UDC table that you specify are valid for that field.

► To attach a user defined code trigger

1. On Data Item Specifications, click the Edit Rule tab.
2. On Edit Rule, click the following option to turn it on:
 - Rule
3. Click the Visual Assist and choose a UDC in the detail area.
4. Tab out of the Rule field and complete the following fields:
 - Product Code
 - Record Type
5. On Data Item Specifications, click OK.

Attaching a Next Number Trigger

The next numbers feature controls the automatic numbering for such items as new general ledger account numbers, voucher numbers, and address numbers. It allows you to specify the numbering system code that you want to use and allows you to automatically increment numbers to reduce transpositions and keying errors.

You use the next number feature when you want the system to enter a default value in a numeric data item if the user does not enter a number. Next numbers are assigned from an array. The combination of system code and index defines how the next number will be assigned.

Next Numbers

The Next Numbers table (F0002) has the following logic:

- 1 record per system and 10-element array

The key to the Next Numbers table (F0002) is system code. The table includes 10 columns for individual next number elements. The system uses each of these elements for a specific hard code within the applications for that system code.

For example, if you specify system code 09 in the Next Numbers program, six rows are populated and four are blank. The system uses each of these coded, populated rows as hard code. The first row defines New Account ID. Within PeopleSoft EnterpriseOne applications that create new accounts, the system retrieves the account number from system 09, row 1 of the Next Numbers table. Row 2 contains Journal Entries. In a master business function (MBF) that creates journal entry documents, the system retrieves the document number from system 09, row 2 of the Next Numbers table.

If you specify system 04 in the Next Numbers program, the system uses a separate set of rows that have hard codes for use within system 04.

- Check digits

The check digits, sometimes called a Modulus 11 check, option allows you to specify whether the system adds a number to the end of each next number that it assigns.

For example, if you are using check digits and the next number is 2, the system adds a check digit such as 7, making the last two numbers 27. Check digits provide a method of randomly incrementing numbers to prevent the assignment of transposed numbers. In the algorithm in the following topic, the system would never assign next number 72 while the check digits feature is activated.

IBM Modulus 11 Self-Check Algorithm

Each position in the base number has a weight factor. Modulus 11 counts positions from the rightmost digit (not including the check digit).

The Modulus 11 weight factors are 2, 3, 4, 5, 6, 7, 2, 3, 4, 5, 6, 7, ... 2, 3, 4, 5, 6, 7, 2 for positions 1 to 31, respectively.

After you set next numbers, do not change them. If you change next numbers, the following might occur:

- System performance will be affected.
- Next numbers functionality will not duplicate numbers; when it reaches the maximum, it will start over.
- You will not be able to change position or add a new entry without modifying the program.

Next numbers connect to the data dictionary. A data item in the data dictionary points to the Next Numbers system. You can access the Next Numbers program (P0002) by typing P0002 in the Fast Path or G00 in the Fast Path and then clicking Next Numbers on the General Systems menu.

► To attach a next number trigger

1. On Data Item Specifications, click the Next Number tab.
2. On Next Number, click the following option:
 - Next Number
3. Complete the following fields:
 - System Code
 - Index

Attaching a Smart Field Trigger

Smart Fields are data dictionary items with attached business functions. The business functions that are attached include named mappings, which simplify the process of choosing a data item with particular functionality. End users do not need to know which business function to use and what parameters to pass; instead, the user simply chooses a data item that has this information. Smart fields can be used in all section types in Report Design. For example, you can use smart fields to derive a column heading or an object value in a tabular section. Smart fields are always in glossary group K.

► **To attach a smart field trigger**

1. On Data Item Specifications, click the Item Specifications tab.
 2. Complete the following required fields:
 - Description
 - System Code
 - Data Type
 - Control Type
 - Row Description
 - Column Title
 - Size
 - File Decimals
 - Display Decimals
 3. Complete the following optional fields:
 - Class
 - Item Occurrences
 4. Click any of the following options:
 - Upper Case Only
 - Row Security
 - Allow Blank Entry
 - Auto Include
 - Do Not Total
 5. From the Form menu, choose Smart Fields.
 6. On Smart Field, complete the following fields:
 - Business Function
Enter the business function that was created for the smart field.
 - Event Name
Specify the event in which the smart field should be triggered.
 - Named mapping
Enter the named mapping that is associated with the business function data structure.
- On Smart Fields, you can click the Browse button to choose a business function. You can also click the visual assist on Named Mapping to choose a value.

Updating the Glossary

Each data item is described within the glossary. The end user of an application sees this description at runtime. Thus, it should explain how the field is used in the application. Within an application, you can access this glossary description in field-level help. You can also do the following:

- Add glossary descriptions for new data items
- Create form-specific or system-specific glossary descriptions
- Create glossary descriptions for different languages
- Customize PeopleSoft EnterpriseOne glossary descriptions

► To update the glossary

1. On Data Item Specifications, click the Item Glossary tab.
2. Enter the glossary information in the media object window.

Adding Glossary Text for Languages

For any data dictionary item, you can add glossary text for different languages. For example, you can create glossary text for the base English language item and also add glossary text for French, Spanish, and German. Glossary text for languages must be added after the data dictionary item has been created.

► To add glossary for an item with languages

1. On Work with Data Items, locate the item that you want to change.
2. From the Row menu, choose Glossary Overrides.
3. On Work With Data Item Glossary Overrides, click Add.
4. On Data Item Glossary Header, complete the following fields, and then click OK:
 - Language
 - Form

Enter a form name if you want the glossary to apply to a specific form only.

If you do not enter a form name, the glossary applies to all forms that use this item.

5. Click OK.
6. On Work with Data Item Glossary Overrides, choose the row that you just added.
7. From the Row menu, choose Glossary.
8. On Data Item Glossary, enter the glossary text that you want to appear.

You can also change the glossary for an item with existing language overrides.

Defining Jargon and Alternate Language Terms

When you create a data dictionary item, you assign descriptions to the row, column, and glossary. Because these descriptions might not offer the flexible terminology that you need, you can assign alternate jargon or language descriptions to each item. Alternate descriptions allow the same data

dictionary item to appear with different row, columns, and glossaries for different users, depending on the system (product) code of the object that they are using.

For example, the cost center field MCU is widely used throughout the system. Its row description is Business Unit, which is a term used by financial applications. However, in distribution applications, this data item appears as Branch/Plant. Likewise, in warehousing applications, the data item appears as Warehouse.

In addition to any alternate terms that you define, users can implement their own language overrides at the application level. The system checks for and resolves overrides in the following order:

1. If a user applies a language override in the application (such as the Form Director Aid or Report Design), the system uses the term indicated by the language override, if one exists.
2. If the user did not specify a language override in the application, then the system determines at runtime whether a system code has been attached to the menu selection. If the menu selection has an attached system code, then the system displays the alternate term dictated by the system code, if one exists.
3. If no alternate term has been indicated for the menu selection, the system determines at runtime whether a system code has been attached to the application. If the application has an attached system code, then the system displays the alternate term that is dictated by the system code, if one exists.
4. If no alternate term has been indicated for the application, then the data dictionary text appears.

In all cases, the system first checks the user's preferred language for an alternate term before checking without language. Language and language overrides always take precedence over nonlanguage overrides. For example, assume that, in an environment in which English is the base language, all the forms have French translations that a user can view by selecting the French override. A form might contain a data dictionary item that in English has an alternate term; in this example, however, the French version of the data dictionary item does not have an alternate term. When it appears in English, the form displays either the main term or the alternate term, as appropriate. When it appears in French, however, the form displays only the main term, even when an alternate term is called for, because the language override takes precedence over displaying the alternate term.

Jargon and alternate language terms must be added after the data dictionary item has been created.

See Also

- ❑ *Updating the Glossary* in the *Development Tools Guide* for information about how to create alternate language glossary items for a data item

► To define jargon

1. On Work With Data Dictionary Items, locate the item for which you want to define jargon.
2. From the Row menu, choose Descript. Overrides.
3. Click Add.

The Data Item Descriptions form appears.

4. On Data Item Descriptions, complete the following fields, and then click OK:
 - Application Override

Enter the system code with which you want to associate the current jargon. Different system codes are associated with specific categories and applications in the system.

- Row Description
 - Column Title
5. Repeat this process for each jargon term that you want to associate with the data item.

► **To update a data item for languages**

1. On Work With Data Dictionary Items, locate the item that you want to update.
2. From the Row menu, choose Descript. Overrides.
3. Click Add.

The Data Item Descriptions form appears.

4. Complete the following fields and click OK:
 - Language
 - Row Description
 - Column Title
5. Repeat this process for each language that you want to associate with the data item.

You can change the row and column text for all applications (interactive or batch) that use a data item.

► **To change row and column text for all applications**

1. On Work With Data Dictionary Items, locate the item that you want to update.
2. From the Row menu, choose Descript. Overrides.
3. On Work With Data Item Descriptions, choose a language record and modify it.

You can change the alpha description by clicking Glossary Overrides and creating a record. You then use the Glossary form to change the description.

Changes to row and column descriptions are not replicated through data replication. To deploy row and column changes to workstations, you must deliver a new full or partial package, or an update package that includes the affected applications. The new or update package deletes the existing row and columns that are stored in a cache on the workstation.

Table Design

A relational database table stores the data that an application uses in columns and rows. Each column is a data item, and each row is a record. You can create one or more tables for use in an application. To create a table, you select data items (the data items must already exist in the data dictionary) to include in the table and assign key fields as indices for retrieving and updating data. You must define your table so that PeopleSoft EnterpriseOne software recognizes that the table exists.

You must use Table Design to generate the table whenever want to do the following:

- Create a new table
- Add or delete a data item
- Add or modify an index

An index identifies records in a table. A primary index identifies unique records in a table. An index is composed of one or more keys, or data items, within the table. An index enables a database management system (DBMS) to sort and locate records quickly.

Adding a Table

Before you add a new table, you should determine whether an existing table contains the data items required by your application. If an existing table does not exist, you must add a new table.

When you add a new table, you must include the following audit trail columns:

- User ID (USER)
- Program ID (PID)
- Machine Key (MKEY)
- Date Updated (UPMJ)
- Time of Day (UPMT)

► To add a table

1. On Object Management Workbench, click Add.
2. On Add J.D. Edwards Object to the Project, click the Table option, and then click OK.
3. On Add Object, complete the following fields and click OK:
 - Object Name
 - Description
 - Product Code
 - Product System Code
 - Column Prefix
 - Object Use

4. On Table Design, click the Summary tab and revise the data in the following fields to alter the table properties:
 - Description
 - Product Code
 - Product System Code
 - Object Use
5. To document the table, click the Attachments tab, and then type the attachment in the text editing area.

Table Naming Conventions

PeopleSoft recommends you use the following naming conventions when you add a table:

The Object Librarian name for a table can be a maximum of 8 characters and should be formatted as follows:

Fxxxxyyy

where the following apply:

F = data table

xx (second and third digits) = the system code, such as:

00 - Foundation environment

01 - Address Book

03 - Accounts Receivable

xx (fourth and fifth digits) = the group type, such as:

01 - Master

02 - Balance

1X - Transaction

yyy (sixth through eighth digits) = object version, such as programs that perform similar functions but vary distinctly in specific processing

LA through LZ - Logical file

JA through JZ - Table join

The column prefix is a two-character code used to uniquely identify table columns. The first character must be alphabetic. The second character can be alphanumeric. You cannot assign special characters (for example, \$, #, or @). The data item name follows the column prefix.

A table description can be no more than 60 characters. Ensure that the table description is the topic of the table. If the table description came from the iSeries, it should be the same name as the file that it represents, such as Address Book Master (F0101) and Item Master (F4101).

Indices

If an index has only one field, list the field as the index name (for example, Address Number).

For coexistence, EnterpriseOne indices must match logicals on the AS/400. When you run the Generate Table command in Table Design, EnterpriseOne automatically determines whether a matching file exists on the AS/400. If a matching AS/400 file does not exist, then ERP creates logical files on the AS/400. If a matching AS/400 file exists, EnterpriseOne does not create any logicals on the AS/400.

If an index has two fields, list them consecutively (for example, Address Number, Line Number ID).

If an index has more than two fields and the first two fields are the same as the first two fields of another index, list the first two fields and follow them by an alpha character (for example, Address Number, Line Number, A). Otherwise, list the fields and follow them by a plus sign (+) (for example, Item Number, Branch, +).

Place a comma and space between each index field and between the last index field and the plus sign. In addition, do not include more than 10 fields in an index.

The total length of the index name cannot exceed 19 characters if the index has two or more fields. If you exceed 19 characters, the compiler displays the warning *Re-definition is not identical....* This situation affects fetches that use the wrong index ID in business functions.

Working with Table Design

Table Design presents the following views within a single window:

- Table Columns, which display the data items that make up your table
- Data Dictionary Browser, which locates data items that can be selected and moved to the Column view
- Indices, which define the unique data items for quicker sorting and updating of the table.
- Properties, which display data item attributes for a selected data item in the Column view. This is a display view only and is primarily used when you define indices.

When you modify or delete data items or indices, you must reconfigure the table. Changes might affect business views and forms that reference that table.

You use Generate to generate a newly modified table. The existing data in the table will be overwritten.

Caution

If you delete a table or delete columns from that table, any business views that reference the table or the deleted table columns will be invalid, and the system will display error messages when you generate the application.

If you use Table Design to delete a table, it deletes only the specifications; it does not delete the physical table.

Choosing Data Items for the Table

Table columns are the data items that store information used by an application. A data item must exist in the data dictionary before you can use it in a table. Tables can contain data items from multiple system codes.

► To choose data items for the table

1. Complete the steps to add a table, or, in Object Management Workbench, choose a table and click the Design button in the center column.
2. On Table Design, click the Design Tools tab, and then click Start Table Design Aid.
3. In the Data Dictionary Browser view, use QBE fields to locate the data dictionary items that you want to include in your table.
4. To include a data item in your table, drag it from the Data Dictionary Browser view to the Table Columns view.
5. To remove a column from a table, choose it and choose Delete from the Edit menu.

After you choose all of the data items for your table, you must define an index.

Defining Indices

You use indices to find specific records and to sort records faster. Table indices are like tabs in a card file. Each index is made up of one or more keys, which are individual data items. You use indices to access data in the most simple manner so you do not have to read the data sequentially.

A table can have multiple indices; however, every table must have only one primary, unique index. The primary, unique index is the one unique identifier for each record in the table. The database should use the index that returns the most detail. It does not always use the primary index. Additionally, you use the primary index to build business views.

See Also

See the following topics in the *Development Tools Guide*:

- ❑ *Performance* for information about performance issues regarding indices
- ❑ *Business View Design* for information about how business views use the primary index from a table

► To define indices

1. On Table Design, arrange the Indices form so that it is active and the Indices menu is visible.
2. From the Indices menu, choose Add New.

You can also drag indices from the column view into the index view.

The index description is Untitled; it is marked with a key that displays the letter P to indicate a primary index.

3. Double-click the index title, enter a name for the index, and then press Enter.
4. On the Table Columns view, choose one or more columns from the columns view and drag them to the index.

A unique index is marked with a single key. You can toggle the unique/not unique status of a key by clicking your right mouse button and choosing Unique from the Index menu. The Unique Primary Index cannot be changed to nonunique status.

5. Right-click the item and choose or deselect the Ascending selection to indicate the ascending or descending sort order for an index column.

An arrow pointing upward indicates that the index column is sorted in ascending order.

Previewing Tables

You can preview how tables will look when you print them.

► To preview your table

On Table Design, arrange the Columns form so that it is active and choose Print Preview from the File menu.

A print preview of your table appears.

Working with Tables

The Object Management Workbench (OMW) provides a central location from which you can manage your tables.

Generating Tables

After you have selected data items and assigned indices for your table, you are ready to configure the table for a specific data source. If you have not established an index, the generation process will fail.

You must generate a table to create the physical table. You cannot add to or update the table until it is generated. Table generation also creates an .H file that is used for compiling in business functions and table event rules.

OMW accesses the Object Configuration Manager application (P986110) to configure tables. You can configure the table within any existing data source. If you do not indicate a data source, the software automatically configures the table for the default map. You can change the path code to generate the table in a different location. Doing so causes the system to perform a drop statement, similar to the remove table, after which the table is re-created. If you regenerate the current table, the data in it will be lost.

You must regenerate a table after you modify it. To ensure that data is not lost, you must export your data, generate the table, and then copy the data back into the table.

► To generate tables

1. On Table Design, click the Table Operations tab, and then click Generate Table.
2. On Generate Table, complete the following fields, and then click OK:

- Data Source
- Password

The system displays a message that indicates whether the generation was successful.

Generating Indexes

If you create additional indices or modify existing ones, you must regenerate them. Doing so modifies the .H file, but you will not lose existing data as you do when you regenerate the entire table.

► To generate indexes

1. On Table Design, click the Table Operations tab, and then click Generate Indexes.
2. On Generate Indexes, complete the following fields, and then click OK:

- Data Source
- Password

Generating Header Files

Occasionally, a table might have no header files. You can generate header files without having to generate the entire table.

► To generate header files

On Table Design, click the Design Tools tab, and then click Generate Header File.

The system generates the header file.

Copying Tables

You can copy tables from one data source to another. Doing so does not copy table specifications. You can also use table conversion to copy tables from one data source to another.

See Also

- *Data Copy with Table Input* in the *Table Conversion Guide* for more information about using the Table Conversion tool to copy tables

► To copy tables

1. On Table Design, click the Table Operations tab, and then click Copy Table.
2. Complete the following fields, and then click OK:
 - Data Source
 - Data Source
 - Object Owner ID
 - Password

Removing Tables from the Database

To completely remove a table from the system, you must use the OMW function entitled Remove Table from Database. If you use Table Design to delete a table, it deletes only the specifications. Table Design cannot physically delete a table.

► To remove tables from the database

1. On Table Design, click the Table Operations tab, and then click Remove Table from Database.
2. On Remove Table, complete the following fields, and then click OK:
 - Data Source
 - Password

Viewing the Data in Tables

To view the data in tables in different databases, you can use the Universal Table Browser. This tool lets you verify the existence of data in a table, as well as to determine the structure of the table. The Universal Table Browser uses JDEBASE APIs to retrieve data from the database, making it independent of the database that you access.

Note

You cannot use EnterpriseOne security to secure users from the Universal Table Browser because the Universal Table Browser is a Windows-executable application, not an EnterpriseOne tools-generated application. However, you can place form security on the Table and Data Source Selection form (W98TAMC). This action secures the Universal Table Browser because the Windows executable cannot function without this form. All column and row security that you set up through Security Workbench applies to the Universal Table Browser.

► To view the data in tables

From Cross Application Development Tools (GH902), choose Universal Table Browser.

1. On Universal Table Browser, choose Open Table from the File menu.
2. Complete the following required fields:

- Table
 - Data Source Name
- Click the following option:
 - Format Data
 - Click OK.

Example: Universal Table Browser (Unformatted Data)

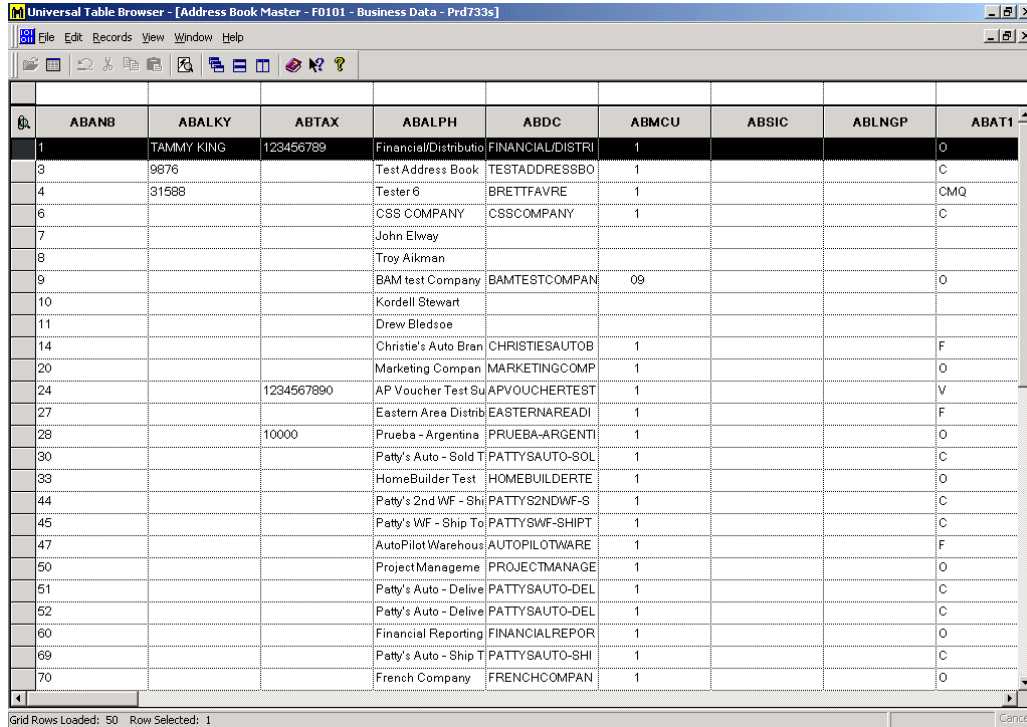
In this example, a database table appears as it was opened with the Format Data option turned off. Note the structure of the information in the ABAN8 column of table F0101. This information is not formatted and appears exactly as it is stored in the database.

	ABAN8	ABALKY	ABTAX	ABALPH	ABDC	ABMCU	ABSIC	ABLNGP	ABAT1
1	TAMMY KING	123456789	Financial/Distrib	FINANCIAL/DISTRI	1				O
3	9876		Test Address Book	TESTADDRESSBO	1				C
4	31588		Tester 6	BRETTFAVRE	1				CMQ
6			CSS COMPANY	CSSCOMPANY	1				C
7			John Elway						
8			Troy Aikman						
9			BAM test Company	BAMTESTCOMPAN	09				O
10			Kordell Stewart						
11			Drew Bledsoe						
14			Christie's Auto Bran	CHRISTIESAUTOB	1				F
20			Marketing Compan	MARKETINGCOMP	1				O
24		1234567890	AP Voucher Test Su	APVOUCHERTEST	1				V
27			Eastern Area Distrib	EASTERNAREADI	1				F
28		10000	Prueba - Argentina	PRUEBA-ARGENTI	1				O
30			Patty's Auto - Sold T	PATTYSAUTO-SOL	1				C
33			HomeBuilder Test	HOMEBUILDERTE	1				O
44			Patty's 2nd WF - Shi	PATTYS2NDWF-S	1				C
45			Patty's WF - Ship To	PATTYSWF-SHIPT	1				C
47			AutoPilot Warehous	AUTOPILOTWARE	1				F
50			ProjectManageme	PROJECTMANAGE	1				O
51			Patty's Auto - Delive	PATTYSAUTO-DEL	1				C
52			Patty's Auto - Delive	PATTYSAUTO-DEL	1				C
60			Financial Reporting	FINANCIALREPOR	1				O
69			Patty's Auto - Ship T	PATTYSAUTO-SHI	1				C
70			French Company	FRENCHCOMPAN	1				O

Grid Rows Loaded: 30 Row Selected: 1

Example: Universal Table Browser (Formatted Data)

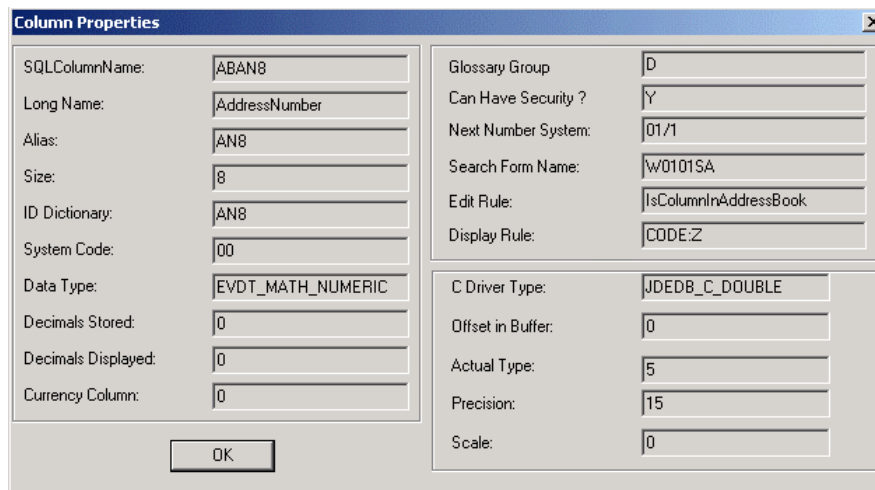
In this example, a database table appears as it was opened with the Format Data option enabled. Notice that the structure of the information in the ABAN8 column of table F0101 is formatted using the data dictionary specifications.



	ABAN8	ABALKY	ABTAX	ABALPH	ABDC	ABMCU	ABSIC	ABLNGP	ABAT1
1	TAMMY KING	123456789	Financial/Distributio	FINANCIAL/DISTRI	1				O
3	9876		Test Address Book	TESTADDRESSBO	1				C
4	31588		Tester 6	BRETTFAVRE	1				CMQ
6			CSS COMPANY	CSSCOMPANY	1				C
7			John Elway						
8			Troy Aikman						
9			BAM test Company	BAMTESTCOMPAN	09				O
10			Kordell Stewart						
11			Drew Bledsoe						
14			Christie's Auto Bran	CHRISTIESAUTOB	1				F
20			Marketing Compan	MARKETINGCOMP	1				O
24		1234567890	AP Voucher Test Su	APVOUCHERTEST	1				V
27			Eastern Area Distrib	EASTERNAREADI	1				F
28		10000	Prueba - Argentina	PRUEBA-ARGENTI	1				O
30			Patty's Auto - Sold T	PATTYSAUTO-SOL	1				C
33			HomeBuilder Test	HOMEBUILDERTE	1				O
44			Patty's 2nd WF - Shi	PATTYS2NDWF-S	1				C
45			Patty's WF - Ship To	PATTYSWF-SHIFT	1				C
47			AutoPilotWarehous	AUTOPILOTWARE	1				F
50			Project Manageme	PROJECTMANAGE	1				O
51			Patty's Auto - Delive	PATTYSAUTO-DEL	1				C
52			Patty's Auto - Delive	PATTYSAUTO-DEL	1				C
60			Financial Reporting	FINANCIALREPOR	1				O
69			Patty's Auto - Ship T	PATTYSAUTO-SHI	1				C
70			French Company	FRENCHCOMPAN	1				O

Example: Column Properties

This example shows the column properties for data dictionary item USEQ. The SQL database name for this item is DTUSEQ.



Column Properties	
SQL Column Name:	ABAN8
Long Name:	AddressNumber
Alias:	AN8
Size:	8
ID Dictionary:	AN8
System Code:	00
Data Type:	EVDI_MATH_NUMERIC
Decimals Stored:	0
Decimals Displayed:	0
Currency Column:	0
Glossary Group	D
Can Have Security ?	Y
Next Number System:	01/1
Search Form Name:	W0101SA
Edit Rule:	IsColumnInAddressBook
Display Rule:	CODE:Z
C Driver Type:	JDEDB_C_DOUBLE
Offset in Buffer:	0
Actual Type:	5
Precision:	15
Scale:	0

Business View Design

A business view is a selection of data items from one or more tables. After you create a table, use Business View Design to choose only the data items that are required for your application. EnterpriseOne uses the business view that you define to generate the appropriate SQL statements necessary to retrieve data from any of the supported databases. After you define a business view, you can create a form that updates data in an interactive application or you can design a report that displays data. Because you choose only those data items that an application requires, less data moves over the network.

Business views are required for creating applications and generating reports; they have the following characteristics:

- Contain some or all of the data items from one or more tables
- Link a PeopleSoft EnterpriseOne application to one or more tables
- Define the data items from multiple tables used by an application (such as table joins or table unions)

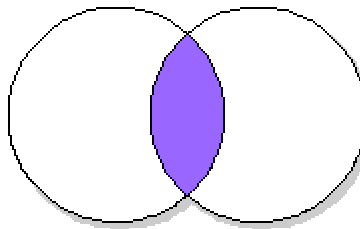
Table Join

Use the table join feature to combine data from individual rows of joined tables. The joining columns satisfy a join condition, such as when the rows have the same values in key columns. The primary table is the table that you are starting from (usually the table on the left in Table Design) and the secondary table is the table that you are going to (usually the table on the right in Table Design). Several types of joins exist, including the following:

- Simple joins, which are also known as inner joins. They include only rows that match both the primary and secondary tables.

The following graphic illustrates a simple table join:

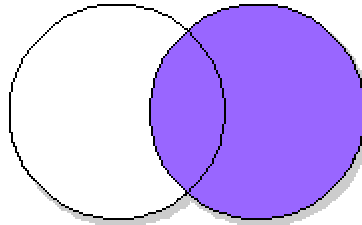
Simple Joins



- Right outer joins, which include rows common to both the primary and secondary tables, and unmatched rows from the secondary table.

The following graphic illustrates a right outer table join:

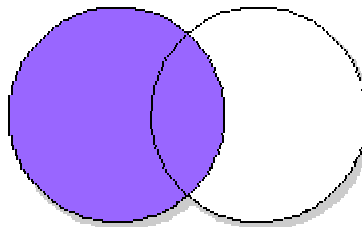
Right Outer Join



- Left outer joins, which include rows common to both the primary and secondary tables, and unmatched rows from the primary table.

The following graphic illustrates a left outer table join:

Left Outer Join



Note About Left Outer Joins and SQL 92 Left Outer Joins

A Left Outer Join is the same as the standard SQL 92 Left Outer Join except that records without a matching right side are always included, ignoring any query against right-side columns.

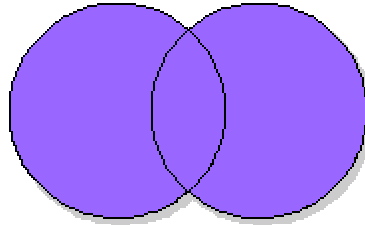
In other words, records with null as values for the right-side columns are always included in the section regardless of any WHERE clause against the right-side columns.

Table Union

A table union joins entire tables. The system first checks for rows from the primary table, and then rows with corresponding columns from the secondary table. If the rows from the two tables contain identical data, then only one of the records is retrieved in the union. Unions include rows from the primary table and corresponding columns from the secondary table.

A union is illustrated below:

Union



Select Distinct

You can use the Select Distinct feature to help eliminate duplicate rows in the business view query.

Indices

You use the primary index from a table to build business views. Business views then carry information from the table to an application. If you need to carry forward additional information other than the primary key to the application, you might need to change the business view index.

Adding a Business View

Before you begin designing a business view, consider the purpose of your application and the data items that it requires, and identify which table or tables contain those data items. Adding a new business view does not affect performance; however, using an existing view that contains more columns than you need might negatively affect performance.

Business views usually contain more fields than are used on the form or grid. If you need a field that is in a business view but not on a form or a grid, you can add the field without changing the business view.

You use different business views for each form type. For forms with grids, the grid should contain fewer fields than the business view to allow for business-specific issues that might affect performance. Typically, search and select forms have the minimum number of items that you need in order to keep them small. They should include only the fields necessary for filtering searches and the associated output fields, such as descriptions.

Find/browse and parent/child forms have more items and are typically larger than search and select forms. Find/browse and parent/child forms should include only those fields needed for filtering and the associated output fields.

Input-capable forms have all of the items from the table and tend to be large. They should include all of the fields necessary to add or update a record, including audit information.

See Also

- ❑ *Performance* in the *Development Tool Guide* for performance information about business views

► To add a business view

1. On Object Management Workbench, click Add.
2. On Add J.D. Edwards Object to the Project, choose the Business View option, and then click OK.
3. On Add Object, complete the following fields and click OK:
 - Object Name
 - Description
 - Product Code
 - Product System Code
 - Object Use

Business View Naming Standards

PeopleSoft recommends that you use the following guidelines when you name a business view:

The Object Management Workbench name for a business view can be no longer than 10 characters and should be formatted as follows:

VzzzzzzzzA

Where:

V = Business view

zzzzzzzz = The characters that represent the primary table

A = A letter that designates the view

For example, V0101A is the first view of table F0101, V0101B is the second view of the same table, V0101C the third, and so on.

External Developer Considerations for Naming Business Functions

External development is the creation of applications by developers who do not work for PeopleSoft, such as Partners in Development and PeopleSoft consultants who create custom applications for specific clients. To prevent interference between PeopleSoft EnterpriseOne and non-PeopleSoft EnterpriseOne objects, you must use caution when naming an external business view. When you create a new business view for an external application, format the business view name as follows:

Vssss9999

Where:

V = Business view

ssss = The system code for the enterprise

9999 = A unique next number or character pattern within the enterprise

Provide up to a 60-character description for a business view. It should reflect the application description followed by the form type, such as Item Master Browse and Item Master Revisions.

Primary unique key fields should remain in the business view. Do not reorganize the primary unique key fields.

Note

At least one business view for each table should include all columns. Use this business view for the level 01 section in all reports upon which the file is based. Also, only one business view is allowed for each form type, except for a header/detail form. For header/detail forms, you can use two business views—one for the header portion of the form and one for the detail portion.

Joined Views

To format the name for a joined view, use the names of the two tables being joined, separated by a forward slash. Place the primary table first.

For example, where F4101 is the primary table in the join view between F4101 and F4102, use the following naming standard:

F4101/F4102

Working with Business View Design

The Business View Design tool displays the following views:

- Table Joins, which defines the tables over which you create the business view
- Available Tables, which locates tables for moving to the Table Joins view
- Selected Columns, which lists the data items from your table that are included in your business view
- Properties

You should generate the business view when you create a new business view or when you add a data item to the business view. In addition, you should be aware of the following when working with business views:

- When you delete a data item from a business view, and that data item is used in an application, an error results when you attempt to run the associated application. If this occurs, you must open the application and delete the item from the application or reselect the column in the current business view to fix the control.
- When you delete an entire table from a business view, any application that uses the business view will not run. If this occurs, you must fix all items that refer to the business view.

- If you delete a business view, any forms that use the business view will fail. If this occurs, you must associate the forms to a new business view and connect all of the controls.

Choosing Tables for a Business View

You must choose one or more tables from which to create the business view. Although you can create a business view over one large table to retrieve and update only those columns that the application needs, performance is diminished when the system retrieves and updates a very large table. Whenever possible, consider joining two smaller tables rather than creating one large table that contains many data items.

► To choose tables for a business view

1. On Object Management Workbench, create a new business view or choose an existing business view, and then click the Design button in the center column.
2. Click the Design Tools tab, and then click Start the Business View Design Aid.
3. On the Available Tables pane of Business View Design, complete the following fields, and then click Find:
 - Description
 - Object Name
 - Product Code
4. Choose one or more tables and drag them to the Table Joins pane.

The pane is called Table Join regardless of whether you are joining multiple tables.

The Table Joins pane displays the tables that you chose, along with the columns that comprise each table. Key symbols appear next to columns that are index keys in the table. The primary table supplies the key to the business view. This is where an application begins a search.

Note

To ensure maximum performance in your application, PeopleSoft EnterpriseOne limits the number of table joins to the following:

- Five tables if all joins are simple joins
 - Three tables if any of the joins is an outer join or in the event of a union
-

5. Designate a table as primary by double-clicking the title bar of the desired table.

This step is optional. If the business view contains multiple tables, the system automatically designates the first table added as the primary table. A crown symbol appears in the upper left corner of the window to indicate the primary table. If a business view contains only one table, that table is the primary table by default.

Note

To delete a table from a business view, choose it and choose Delete from the Table menu. Alternatively, right-click the table and choose Delete from the pop-up menu.

Choosing Data Items for a Business View

After you choose one or more tables for the business view and indicate the primary table, you must choose the data items to include in the business view. All data items in the primary table, along with any tables to which the primary table is joined, are available for the business view.

Choose the data items that you want to use in an interactive application or batch job. When you create an application, you do not have to use every item in the business view.

Important Notes Regarding Data Item Selection

- If you have two tables in a join, the primary index is automatically selected for both. Index keys of selected tables are automatically highlighted for use in the business view. Except for the primary table index keys, you can remove index keys if they are not going to be used in the business view. You cannot remove index keys for the primary table from the business view.
 - If you create a join, items from both tables are automatically selected. If the same data item appears in multiple tables, you need to select the data item from only one table.
 - You cannot join unlike items.
 - Do not select the same data item from two different tables because doing so causes the data item to appear twice in the business view.
-

► To choose data items for a business view

1. On the Table Joins pane of Business View Design, double-click the data items to include in the business view.

Selected data items appear highlighted in the Table Joins window. As you choose each item, the system adds it to the Selected Columns window.

Caution

Do not select more than 256 columns for your business view. An application that uses a business view that contains more than 256 columns will fail at runtime.

2. To delete a data item from your business view, double-click the item in the Table Joins window to remove it from the business view.

Using Select Distinct

If a business view includes the primary key of the primary table (default implementation), then every row of the business view query is unique. The primary key has a different value in each row of the primary table. If the business view does not have a primary key in the primary table, then duplicate rows can occur during the business view query. You can eliminate duplicate rows in the business view query by using the Select Distinct feature while designing a business view.

For example, Journal Entry is unique by line number within a document. You need only the first document number with Line 1, not all line numbers within the document. The Select Distinct feature fetches only the first occurrence of the document number, not all the line numbers within it.

Any business view with a primary table that contains one or more of the following columns, which are used for currency support or security features, might cause the Select Distinct feature to output duplicate values:

CO	Company
CRCD	Currency Code - From
CRDC	Currency Code - To
CRCX	Currency Code - Denominated In
CRCA	Currency Code - A/B Amounts
LT	Ledger Type
AID	Account ID
MCU	Business Unit
KCOO	Order Company (Company Code)
EMCU	Business Unit Header
MMCU	Branch
AN8	Address Number

► To use Select Distinct

1. On Business View Design Aid, choose the primary table of your view.
2. Choose Distinct Mode from the Table menu.
3. From the Table menu, choose Change Index to change the index of the primary table to a non-unique index.

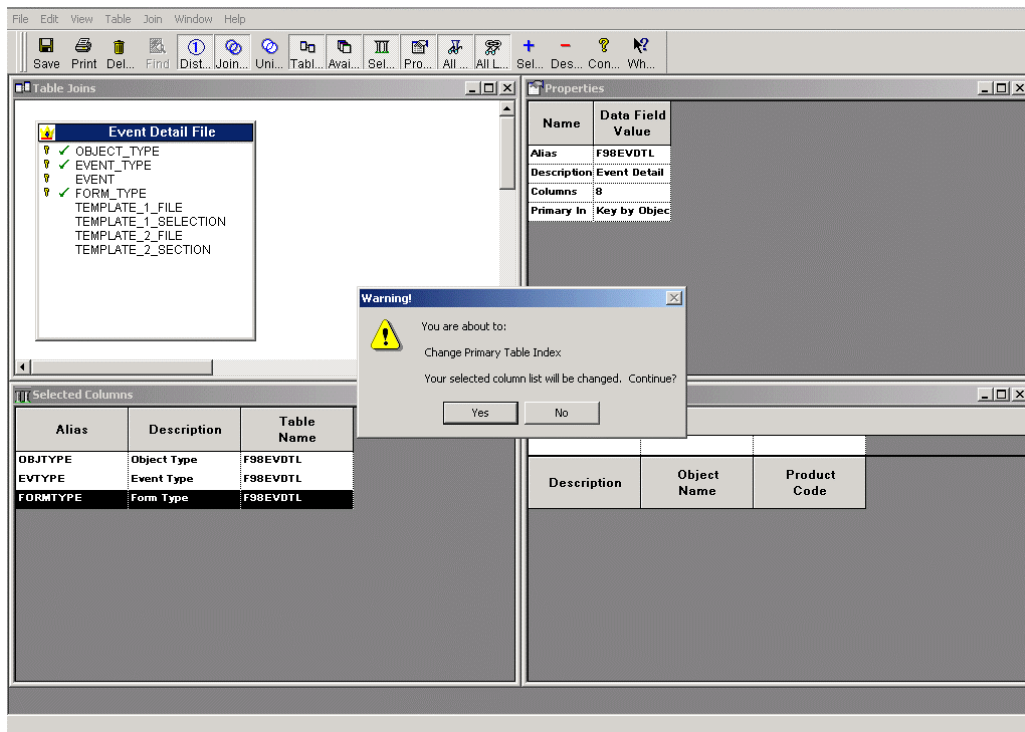
Example: Select Distinct Feature

The business view used for this example, Event Detail Business View (V98EVDTL), uses the primary index of the primary table by default; in this example, it is the Event Detail File table

(F98EVDTL). The primary index of an EnterpriseOne table must be unique. A unique primary index ensures that the system does not return duplicate values when the business view query is generated. Business View Design Aid also allows you to use any other index of the primary table when you process the business view. The following steps illustrate how the Select Distinct feature works.

4. In Business View Design Aid, choose the primary table.
5. From the Table menu, choose Change Index to change the primary table index.

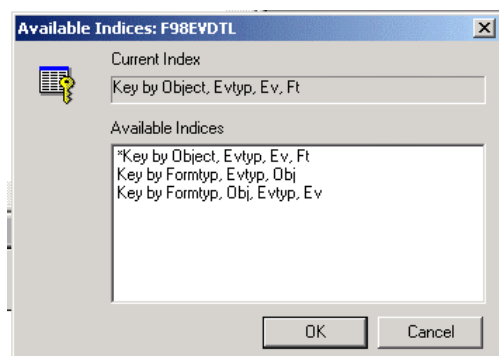
Change Index is available only for the primary table.



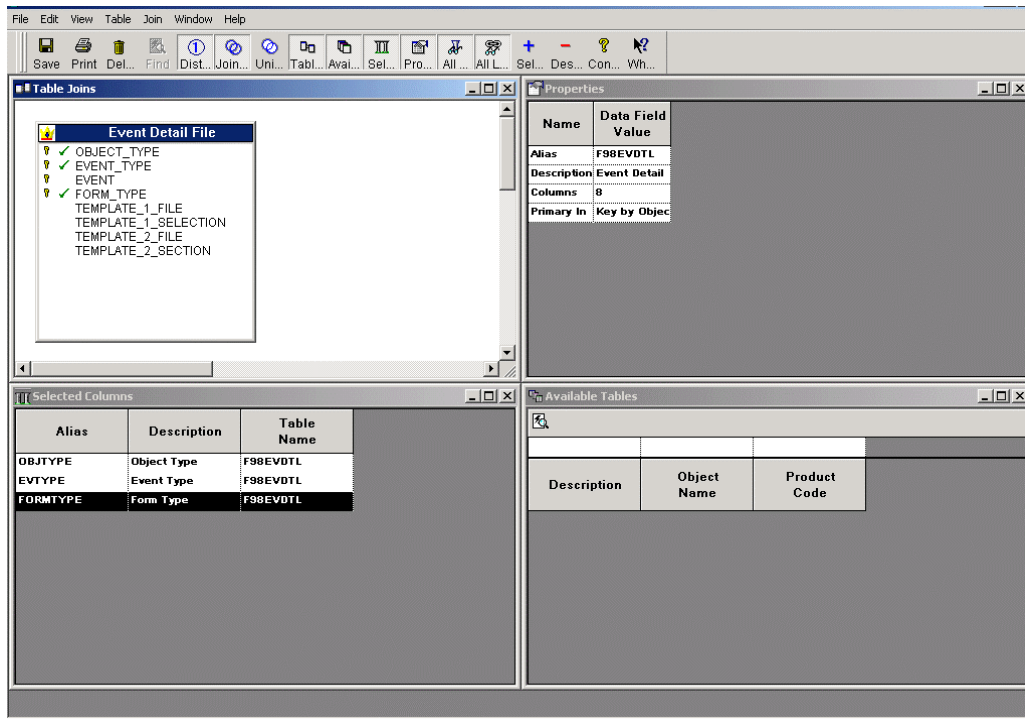
The system displays a warning, indicating that your selected column list will be changed.

6. Click Yes to continue.

The Available Indices form appears. The first edit field on the form displays the current index of the table used by the business view. The default is the primary index.



7. For this example, choose Key by Formtyp, Evtype, Obj from Available Indices, and then click OK.



The Table Joins list and Selected Columns list change to reflect the keys of the new index.

8. Save the changes and exit Business View Design Aid.

Now, if you run an application that uses the V98EVDTL business view with Select Distinct off and the changed business view index (Key by Formtyp, Evtyp, Obj), the generated SQL statement is:

```
SELECT EDOBJTYPE, EDEVTYPE, EDFORMTYPE FROM PVC. F98EVDTL
```

Using this example, you might now have 281 rows of data from table F98EVDTL.

9. Reopen the V98EVDTL business view.
10. From the File menu, choose Select Distinct.
11. Choose Change Index to re-select the Key by Formtyp, Evtyp, Obj index from Available Indices and then click OK.
12. Save the business view and exit Business View Design Aid.

You might need to exit and restart the software. Because the software stores the business view in cache memory, even though you have changed the business view, the previous business view will run until it is cleared from cache memory.

Generate and rerun the same application using the V98EVDTL business view that you just completed (with Select Distinct on and the changed business view index Key by Formtyp, Evtyp, Obj). The generated SQL statement is:

```
SELECT DISTINCT EDOBJTYPE, EDEVTYPE, EDFORMTYPE FROM PVC.
F98EVDTL
```

Using this example, you might now have only 53 rows of data from table F98EVDTL.

Creating a Table Join

Create table joins to access multiple tables in a single application.

You typically use joins for non-input-capable forms, such as find/browse forms, and reports. You do not usually use them for forms that update and add to the database because the relationship between the records must be precise. Use joins for input capable forms when the relationship between two tables is simple. Joins are faster than using individual fetches to retrieve data.

If a business view uses multiple tables, link them by establishing joins between columns in those tables. The links indicate how rows between a table correspond to rows in another table.

Data item attributes are defined in the data dictionary. Use the Properties form to view attributes for a column when determining whether you can use it in a join. The highlighted data item in the Selected Columns form displays the properties of that data item in the Properties form.

Review each table and decide how the data in each table is related to the data in other tables for your application or report. You might need to add columns, build indexes, or even create new tables. If you build new indexes, consider your needs carefully before you do so.

► To create a table join

1. In Business View Design, click and draw a line that links a column in the primary table to a column in a related table.

You can create table joins only between similar columns. The names of the columns can be different, but the attributes for Data Type and Decimals must be identical. To determine whether data items are candidates for a join, click a data item in the Table Join pane and view the data item attributes that appear in the Hover Hints for each data item.

2. Click the line that joins the two tables to focus on it.
3. To delete a join, choose it and then choose Delete from the Join menu. Alternatively, right-click the join and choose Delete from the pop-up menu.
4. From the Join menu, choose Types and then choose one of the following join types:
 - Simple
Simple is the default join type.
 - Left Outer
 - Right Outer
5. From the Join menu, choose Operators and then choose one of the following operators:
 - Equal (=)
Equal (=) is the default operator.
 - Less than (<)
 - Greater than (>)
 - Less or equal (<=)
 - Greater or equal (>=)
6. Click the Selected Columns view to sequence the columns.

Creating a Table Union

Unions are used to pull rows from tables that have the same structure. Unions pull rows that exist in either table.

► To create a table union

1. On Business View Design, choose Union Mode from the Table menu.

Alternatively, use the appropriate icon on the toolbar.

The view indicates Table Joins.

2. Choose the tables for which you want to create a table union.

Data Structures

Data structures are a key element of any programming language or environment. A data structure is a list of parameters that passes data among applications and tables or forms. PeopleSoft EnterpriseOne uses data structures in the following instances:

- The system generates a data structure.
- You create a data structure.

System-Generated Data Structures

The two types of system-generated data structures are as follows:

- Form
- Report

Form Data Structures

Each form with an attached business view has a default data structure. Data structures receive parameters from or send parameters to other forms during Form Interconnects. You maintain the data structure by using the Form/Data Structure menu option in Form Design.

Report Data Structures

A batch application with an attached business view can receive parameters from or send parameters to a data structure. You can create and maintain the data structure from the Report/Data Structure menu option in Report Design. Unlike a form data structure, this type of data structure is not automatically populated with data items.

User-Generated Data Structures

As a user, you can create three types of data structures, as follows:

- Media object data structures
- Processing options data structures
- Business function data structures

Media Object Data Structures

To enable an application for media objects, you must create a data structure to pass arguments from the application table to the media object table. To work with a data structure for media objects, create a GT object type, or select an existing one to modify in Object Management Workbench.

Processing Options Data Structures

You use processing options to create an input property sheet. You use a parameter list to pass processing options to an application. You can create a processing option data structure template or modify an existing template in Object Management Workbench.

Business Function Data Structures

Any business function, whether it uses C or Business Function Event Rules as its source language, must have a defined data structure to send or receive parameters to or from applications. You can create a DSTR object type, or choose an existing object type to work with in Object Management Workbench. You can also create data structures for text substitution messages. Additionally, you can attach notes, such as an explanation of use, to any data structure or data item within the structure.

See Also

- ❑ *Messaging in the Development Tools Guide* for more information about text substitution messages

Working with Interconnection Data Structures

The Form Design or Report Design tool, not Object Management Workbench, maintains interconnection data structures. The data structure enables an application or report to pass values between interconnection forms or sections.

For interactive applications, the default data structure contains only the keys from the business view selected for a form. If you want to pass a value for a data item that is not in the default data structure, you must add the data item to the data structure.

The system creates a default data structure at the report level for a report with an attached business view. The default data structure is empty. If you add data items to the data structure, both the structure and the report itself is maintained by Report Design.

Interconnecting Forms

You create the default data structure by using keys from the business view selected for the form. You can access the form data structure from Form Design. From the Form menu, you can choose Data Structure to display the Form Data Structures form.

The items in the existing data structure appear in the Structure Members list. Two tabs appear for modifying the data structure. The Available Objects tab lists data items in the attached business view. If you want to add an item to the data structure that is not in the Available Objects list, then you can select it from the Dictionary Items list. The Dictionary Items tab accesses items from the Data Dictionary.

► To change form data structures

1. On Form Design Aid, focus on the form with the data structure that you want to modify, and then choose Data Structure from the Form menu.

The Form Data Structure form appears.

2. To add an object from the attached business view or grid column, click the Available Objects tab, and then drag the object into Structure Members on the left side of the form.
3. To add a specific data dictionary item, complete the following:
 - a. Click the Dictionary Items tab.
 - b. In the QBE row, complete any of the following fields, and then click Find.
 - Alias
 - Data Item
 - Description
 - Product Code
 - Data Type
 - Item Size
 - c. Click the data item in the detail area.
 - d. Drag the desired data dictionary item into Structure Members on the left side of the form.
4. To remove an object from the data structure, choose the item in Structure Members, and then click Delete.
5. Click OK.

Changing Report Data Structures

The system creates an empty default data structure for any report section with an attached business view. This data structure receives values from another report in a report interconnection. If a required item is not in the existing data structure, you can choose it from the data dictionary and include it in the Structure Members list. To modify a report data structure, use Report Design and follow the steps for changing a form data structure.

Example: Changing Interconnection Data Structures

The following example describes a situation in which you might want to change an interconnect data structure.

Assume that you have created two fix/inspect forms, both of which use the same business view. The second form shows more detail, such as additional columns from the same record. If you do not want to refetch the record in the second form, you should change the data structure for the second form.

The data structure for the second form should contain all of the information (fields on the form) and data items in the data structure, in addition to the keys in the business view. In this case, you should set the form options to *No fetch on form business view* for that form. This also applies if you are not updating the record with new information on the second form, but you are passing that information back to the first form and allowing the first form update the database. In this case, you should turn off the *Update on form business view* option.

Depending on your configuration, you might want to fetch the entire record once in the first form and pass it to the second form, or let the second form refetch the record. In either case, you will have two separate business views, with different columns from the same record. If the second form is a form that will not be accessed often, do not fetch the fields in the business view of the first form. For the first form, use a business view that has only the columns needed for that form. If you fetch all of the columns needed for both forms, you perform only one *Select* from the database and all the data is passed over the network once, instead of having two separate *Select* statements generated by two different business views traversing the network. If hardware or memory limitations for the workstation do not prevent you from doing so, then fetch all columns for that record for both forms and allow that information to reside in the workstation's memory.

Displaying Type Definitions

You can display the type definition for a data structure. The type definition is stored in the clipboard. You can paste it into another application, such as a word processing application, to save or review the definition.

Creating a Data Structure

A data structure object is required to create a business function. A data structure is used to pass data between an application and a business function. EnterpriseOne Tools allows you to create an encapsulated data structure object. You can create several types of data structures that are maintained independently as objects in Object Management Workbench.

Creating a Business Function Data Structure

Business function data structures are used by C business functions and business function event rules to pass data between the business functions or application event rules.

► To create a business function data structure

1. On Object Management Workbench, click Add.
2. On Add J.D. Edwards Object to the Project, choose the Data Structure option, and then click OK.
3. On Add Object, complete the following fields:

- Object Name

The naming convention for business function data structures is as follows:

DxxxxyyyyA

Where:

D = Data structure

xxxx = The system code

yyyy = A next number

A = A multiple data structure differentiator (A, B, C, and so forth) used when a function has more than one data structure

- Description
- Product Code
- Product System Code
- Object Use

4. Choose the following option and click OK.

- Regular Data Structure

See Also

- *Understanding PeopleSoft EnterpriseOne Naming Conventions in the Development Guidelines for Application Design Guide*

Creating a Media Object Data Structure

A special data structure object is required to work with media objects. The Media Objects storage table (F00165) contains information used to determine how to access specific media object attachments. This information is stored in the Object Name and Media Object Key (GDTXKY) fields. For example, in the Sales Order application, GDTXKY stores the key for each sales order that is entered. The F00165 table contains the following:

- Object Name, such as GT4201A
- GDTXKY (Key), such as sales order number|order type|company
- GDTXVC (Media Object Attachment)

The media object key contains the actual key to the location in which a specific record in the application is stored. This key typically is the same as the keys to the table that is used by the grid or form. The media object key must have a unique value for each media object attachment so that the system retrieves the correct attachment.

► To create a media object data structure

1. On Object Management Workbench, click Add.
2. On Add J.D. Edwards Object to the Project, choose the Media Object Data Structure option and click OK.

The Add Object form appears.

3. On Add Object, complete the following fields and click OK:

- Object Name

The PeopleSoft EnterpriseOne naming convention for a media object data structure is as follows:

GtxxxyyA

Where:

GT = Media object

xxxx = The table name, excluding the letter F

yy = A next number

A = A letter that differentiates multiple media objects if the table has multiple media objects

- Description
Describe the subject of the media object in no more than 60 characters.
 - Product Code
 - Product System Code
 - Object Use
4. On Data Structure Design, click the Design Tools tab, and then click Data Structure Design.
 5. To add a specific data dictionary item, click the Dictionary Items tab.
 - a. In the QBE row, complete any of the following fields, and then click Find:
 - Alias
 - Data Item
 - Description
 - Product Code
 - Data Type
 - Item Size
 - b. Click the data item in the detail area.
 - c. Drag the desired data dictionary item into Structure Members on the left portion of the form.
 6. To change the Structure Member Name, double-click the row in the detail area and enter a new name.

Note

Use Hungarian notation rules for naming data structures.

7. Click OK.

Creating a Processing Options Data Structure

Processing options are used to create an input property sheet.

► To create a processing options data structure

1. On Object Management Workbench, click Add.
2. On Add J.D. Edwards Object to the Project, choose the Data Structure option, and then click OK.
3. On Add Object, complete the following fields:

- Object Name

The EnterpriseOne naming convention for processing option data structures is as follows:

Txxxxyyyyy

Where:

T = Processing option data structure

xxxxyyyyy = The program number for the application or report

- Description
 - Product Code
 - Product System Code
 - Object Use
4. Choose the following option and then click OK:
 - Processing Option Template

Defining a Data Structure

After you create a data structure, you must define the data objects to include in the data structure. You can modify an existing data structure by adding new data objects to it or deleting data objects from it.

See Also

- *Processing Options* in the *Development Tools Guide* documentation for information about working with processing options and their data structures and templates

Designing a Data Structure

After you create a data structure with OMW, use the following process to specify how it will function.

► To design a data structure

1. On Object Management Workbench, check out the data structure with which you want to work.
2. Choose the data structure, and then click the design button in the center column.
3. On Data Structure Design, click the Design Tools tab, and then click Data Structure Design.
 - a. To add a specific data dictionary item, complete the following:
 - i. Click the Dictionary Items tab.

- ii. In the QBE row, complete any of the following fields, and then click Find.
 - Alias
 - Data Item
 - Description
 - Product Code
 - Data Type
 - Item Size
 - c. Click the data item in the detail area.
 - d. Drag the desired data dictionary item into Structure Members on the left side of the form.
4. In the Structure Members area, to change the structure member name, double-click the row in the detail area and enter a new name.

Note Concerning Data Structure Naming Conventions

Use Hungarian notation rules.

If you are unsure about which data dictionary item you want, click Data Dictionary to search for data dictionary items and then view details about the data items that you select.

You can view detailed information about a data item in the data structure by choosing it, and then clicking Data Dictionary Detail.

5. In the Structure Members area, you can indicate the required status by leaving the Required Member/Req option blank or choosing one of the following:
 - Choose X to indicate that the field is optional.
 - Choose the checkmark to indicate that the field is required.
6. Choose a direction for the data flow by clicking the arrow for the Input/Output/IO option until it changes to the appropriate direction and type.
7. To remove an object from the data structure, select the item in Structure Members and then click Delete.
8. To define attachments for the data structure, click Data Structure Attachments.
9. To define attachments for a data structure item, click Data Structure Item Attachments.
10. To launch the Cross Reference utility, click Cross Reference.
11. When you are finished, click OK.

Creating a Type Definition

After you design a data structure, you create a type definition, which is a C code representation of a data structure definition used in business functions. It is stored in the clipboard after you create it so that you can then paste it into the appropriate section of the .h file. This process saves you time because you do not have to type the code in the .h file.

► **To create a type definition**

1. On Data Structure Design, click the Design Tools tab.
2. Click the Create a type definition button.

Your type definition is stored in the clipboard. You can now paste it into the appropriate section of the .h file.

Cross Reference Facility

You can use the Cross Reference Facility to determine where and how specific kinds of objects are used. You can also view relationships between objects and their components. For example, you can do the following:

- Identify each instance in which a business function is used
- View a list of forms within an application
- Display all fields within a business view or form interconnection
- Cross-reference all applications in which a specific field, event rule, or control is used

You can rebuild cross-reference relationships when you change an object or a component.

Searching for Objects

You can search for objects by search type and object name.

► **To search for objects**

From the Cross Application Development Tools menu (GH902), choose Cross Reference Facility.

Click one of the following tabs:

- Data Items
- Interactive Applications
- Batch Applications (the batch application object type in Cross Reference is UBE)
- Business Functions
- Business Views
- Data Structures
- Tables
- Forms

Searching for Data Items

You can determine where specific data items are used. You can search for the following:

- Forms that use a data item
- UBEs that use a data item

- UBE event rules that use a data item as a variable
- Applications that use a data item as a variable
- Named event rules that use a data item as a variable
- Functions called by smart field data items
- Edit rule functions called by a data item
- Display rule functions called by a data item
- Search forms used by data items
- Processing options that use a data item
- Generic text data structures that use a data item
- Business function data structures that use a data item
- All data structures that use a data item
- Tables that use a data item
- Indices that use a data item
- Business views that use a data item
- Table event rules that use a data item

Searching for Interactive Applications

You can locate a variety of information about interactive applications. You can search for the following:

- Applications that call an application
- Applications that call a business function
- Data structures that are used by an application
- Tables that are used by an application
- Forms for an application
- Data items for an application
- Data items that are used as variables in an application
- Processing options for an application
- Business views for an application

Searching for Batch Applications

You can locate information about batch applications (often referred to as UBEs) and how they are used. You can search for the following:

- UBEs that call a business function
- Tables that are used by a UBE
- Data structures that are used by a UBE

- UBEs that are called by a UBE
- UBEs that call a UBE
- Processing options for a UBE
- Business views for a UBE
- Data items that are used by a UBE
- Data items that are used as variables in a UBE

Searching for Business Functions

You can locate a variety of information about business functions and how they are used. You can search for the following:

- The places in which a business function is used
- Data structures that are used by a named event rule
- Business functions that are called by an application
- Business functions that are called by a UBE
- Tables that are used by a business function
- Business functions that are called by a business function
- Business functions that call a business function
- Data items that are used as variables in a named event rule
- Tables that are used by a named event rule
- Business functions that are called by a named event rule
- Smart field data items that call a column header function
- The places in which a function is used
- Smart field data items that call a value business function
- Data items that call edit rule functions
- Data items that call display rule functions
- Source and header files by string
- Table event rules that use a business function

Searching for Business Views

You can locate information about business views and how they are used. You can search for the following:

- Applications that use a business view
- UBEs that use a business view
- Forms that use a business view
- Data items for a business view

- Tables for a business view

Searching for Data Structures

You can locate a variety of information about data structures and how they are used. You can search for the following:

- Applications that use a data structure
- Business functions that use a data structure
- UBEs that use a data structure
- Data items for a processing option
- Data items for generic text data structures
- Data items for a business function data structure
- Data items for all data structures
- Table event rules that use a data structure

Searching for Tables

You can locate a variety of information about tables and how they are used. You can search for the following:

- Business functions that use a table
- Applications that use a table
- UBEs that use a table
- Named event rules that use a table
- Forms that use a table
- Data items for a table
- Indices for a table
- Business views that use a table
- Data items that are used by table event rules
- Business functions that are used by table event rules
- Data structures that are used by table event rules
- Tables that are used by table event rules

Searching for Forms

You can locate a variety of information about forms and how they are used. You can search for the following:

- Forms that are called by an application
- Tables for a form

- Data items for a form
- Business views for a form
- Data items that use a search form

Searching for Event Rules

You search for event rules to help you find an existing event that you can use.

► To search for event rules

1. On the Cross Reference Facility search form that you have chosen, choose Event Rules from the Form menu.
2. On ER Search, complete the following fields and click Find:
 - Object Name
 - Form Name
 - Data Field Name

Viewing Field Relationships

The Field Relationships form is meaningful only for the following cross-reference search types:

DA Data items that are used by an application

FA Forms for an application

FI Forms that use a data item

SA Data structure for an application

In these instances, the Field Relationships form displays the control type for a field, such as one of the following:

- BC for a business view column
- FI for a form interconnect control
- GC for a grid control
- FC for a form control

► To view field relationships

On the Cross Reference search form that you chose, choose Field Relationships from the Form menu.

1. On Field Relationships, complete the following fields as necessary:
 - Object
 - Form Name

- Field
- Control

The control field does not have a Search button. The possible control types are as follows:

- BC - Business View Field
- PO – Processing Option
- FI - Form Interconnect
- VAR - Variable
- FC – Form Control
- FCW - Form Control Work field
- GC - Grid Column
- GCW - Grid Column Work field

2. Click Find.

All of the controls that appear on a given application or form appear in the detail area.

Rebuilding Cross Reference Information

When developers modify objects, cross-reference information might become out-of-date with objects in the system. Because the cross-reference files are not automatically rebuilt when objects are modified, you must do so at the time of inquiry. You can also regularly schedule cross-reference builds to ensure that the cross-reference information is updated regularly.

View the date on which a record was built in the far right column of the grid on each cross-reference form of the Cross Reference utility. If information is out-of-date, use the Rebuild option from any of the cross-reference forms.

Cross-reference builds use relational database tables, not local specifications.

► To rebuild cross reference information

1. On Cross Reference, choose Rebuild Relationships from the Form menu.
2. Choose the objects that you want to rebuild.

Note

The rebuild process can take several minutes.

Event Rules

This topic discusses the use of event rules and logic and describes how to use BrowsER.

Event Rules Design

Use Event Rules Design to create business logic for an application. You can create event rules that do the following:

- Perform a mathematical calculation
- Pass data from a field on a form to a field on another form
- Count grid rows that are populated with data
- Interconnect two forms
- Hide or display a control using a system function
- Evaluate If/While and Else conditions
- Assign a value or an expression to a field
- Create variables or programmer-defined fields at runtime
- Perform a batch process upon completion of an interactive application
- Process table input and output, validate data, and retrieve records

Prerequisites

- ❑ Create an application with one or more forms.
- ❑ Understand the difference between database items and data dictionary items.
- ❑ Understand the relationship between controls, events, and event rules.
- ❑ Determine the purpose of each form used in the application.
- ❑ Answer the following questions:
 - What logic is required?
 - For which control are you creating logic?
 - For which event will the logic occur?
 - Which runtime structures are affected?

Controls

A control is a reusable object that appears on a form. Examples include push buttons, edit fields, and grids. A form itself is also considered a control.

Controls can be simple or complex. Simple controls have few event points to which logic can be attached. Complex controls can have many event points to which logic can be attached.

Events

Events are activities that occur on a form, such as entering information a form or exiting a field by using the Tab key. Events can be initiated by the user or the application. A single control might initiate multiple events. The system also initiates some events, such as *Last Grid Record Read*, when certain actions occur.

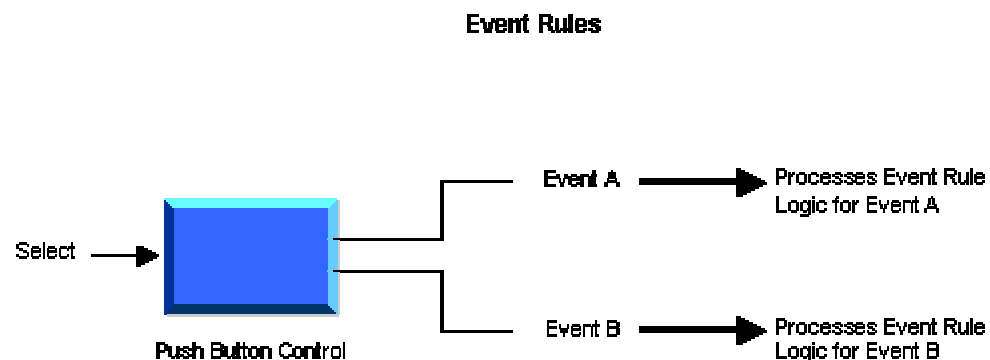
Form Processing

Form processing refers to the business logic associated with each form. By default, each type of EnterpriseOne form automatically processes various events. You specify additional logic by using Event Rules Design. Form processing depends on the occurrence of specific events, such as initializing a form or changing the value of a field.

Event Rules

Event rules are logic statements that you can create and attach to events. PeopleSoft EnterpriseOne software uses two types of event rules: business function event rules and embedded event rules. Event rules are initiated when events occur at runtime.

The following illustration shows this relationship:



You can attach multiple event rules to one event. The various kinds of event rules include:

- Conditional statements, such as If/Else/End If
- While loops
- Assignments to statements
- Calls to encapsulated functions

- Form or report interconnections
- Calls to system functions
- Table I/O operations

Business Function Event Rules

Business function event rules are encapsulated, reusable, business logic that you create using Event Rules Design, rather than C programming. Business function event rules are stored as objects and are compiled. Business function event rules are sometimes called Named Event Rules (NERs).

Embedded Event Rules

Embedded event rules are specific to a particular table, interactive application, or batch application. They are not reusable. Examples include using form-to-form calls, hiding a field that is based on a value in a processing option, and calling a business function. Embedded event rules can be in application event rules (interactive or batch) or in table event rules. They can be compiled or uncompiled.

Application Event Rules

You can add business logic that is specific to a particular application. Interactive applications connect event rules via Form Design, while batch event rules use Report Design.

Table Event Rules

You can create *database triggers*, or rules that you attach to a table by using Table Design Event Rules. The logic that is attached to a table is run whenever any application initiates that database event. For example, to maintain referential integrity, you might attach rules to a master table that delete all children when a parent is deleted. Any application that deletes information from that table does not need to have the parent/child logic embedded in it because that logic exists in the table.

Event Rule Buttons

The Event Rules Design form displays the following buttons for generating different types of business logic:

Business Function	Attaches an existing business function.
System Function	Attaches an existing EnterpriseOne system function.
If/While	Creates an If/While conditional statement.
Assignment	Creates an assignment or a complex expression.
Report Interconnect	Establishes a connection to a batch application or report.
Form Interconnect	Establishes a form interconnection.
Else	Inserts an Else clause, which is valid only within the bounds of If and End If.

Variable	Creates a programmer-defined field, which has data dictionary characteristics for application-specific purposes but does not reside in the data dictionary.
Table I/O	Allows event rule support for database access. Performs table input and output, data validations, and record retrieval.

Event Rules Based on Form Type

Each form type has specific event rules that you typically use for that form type, as indicated by the following table:

Form Type	Event Rules
Find/browse	Form-level and grid level
Fix/inspect	Form-level only
Header detail and headerless detail	Form-level and grid level

Runtime Processing

Runtime processing refers to how, at runtime, the system evaluates various events (such as initializing a form, clicking a button, and using the Tab key to move between fields) and their attached event rule logic. Event rule logic is attached to events, which, in turn, are attached to controls.

Form Design provides several different form types, each of which includes predefined fields and features that are specific to the form type. For example, a find/browse form automatically includes a Find menu option or toolbar button with appropriate functions attached to it. When you type search text in a filter field or Query by Example (QBE) field, and then click the Find button on the toolbar, the runtime engine processes logic to fetch a record.

To avoid generating unnecessary event rules, you should understand the different field types and associated capabilities that characterize each form type. You can also customize existing event rules.

Runtime Data Structures

Runtime data structures are structures or blocks of memory that hold data in memory as the system reads, processes, and writes the data to the database. You should know what is happening to each form at runtime. You should know what is in a runtime structure at a given event point.

The runtime system dynamically creates runtime data structures. For example, if a form contains hidden controls, the system allocates memory for those controls even though they are not visible on the form. When you pass processing option values in a form, the system allocates memory to store the processing option value.

Available Objects and Runtime Data Structures

An available object is represented by a two-character, alphabetical code that characterizes the source of data and determines how the object data is used in an interactive application at runtime. Available objects make up the runtime data structure for a form.

During runtime processing, the system stores data in memory in an internal data structure. A data structure enables data to pass to another field on the same form or between forms. Certain fields of the data structure temporarily store data during runtime. When the data is no longer needed, it is deleted so that the system can process another record.

Available objects include following:

- BC** A column in the business view. Business view columns for both the form view and the grid view appear in this list. The system fills these columns with values from the database when it performs a fetch. The system writes the values to the database during an add or update.
- GC** A column in the grid. The row that the value references depends on which event is accessing the GC. During the fetch cycle, the system fetches the row. It is usually the selected row. In some circumstances, grid column objects also denote a particular physical column in the grid instead of a value. An example is the Set Grid Line Font system function.
- GB** The grid buffer. This buffer is one row of data space that is independent of the lines that the system reads from the database and writes to the grid. The grid buffer allows you to manipulate column data for a line that you want to insert or update without affecting the grid's present state. You access the grid buffer through the available object Grid Buffer Column (GB), which appears after the Grid Columns (GC) in the list of available objects in event rules. Each grid contains only one instance of each GB column. The grid buffer is not associated with any particular line.
- FC** A control on the form. If the control is a database item, this field corresponds to a BC object. Furthermore, if the control is not a filter, the FC object represents the same value as the BC object, and changing one of these results in changing both.
- FI** A value passed through a form interconnection. You access this object either to read values that are passed into the form or to set values to be passed back. These objects correspond to the elements of the form data structure.
- PO** A value passed from a processing option. These values are passed into the application when a user launches it. Any form in that application can access them. Processing options can either be entered by the user, or they can be set up in a particular version of an application.
- QC** A column from the Query by Example (QBE) line in the grid. These objects represent the values in any QBE cell on the grid. They include wildcards, but do not include any comparison operators. Likewise, assignments to these objects can include wildcards, but not comparison operators. You can use a system function to set comparisons. QC objects do not affect the data selection on an update grid.
- HC** Hypercontrol item. These objects represent hypercontrol items on the form's hypercontrol. They can be used to enable and disable those items on the form menu and on the toolbar, as well as invoke them through event rules.
- VA** Event rules variables. These objects represent any variables that the developer set up in event rules. The system does not manipulate event rules.
- SV** System variables. These objects represent some environment variables that are accessible to event rules.

SL	System literals. These objects represent some constant system values that are accessible to event rules.
TP	Tab page object.
TK	A column in the table that contains the table event rule.
CO	A constant, such as the return code for an error.
TV	Text variable.
RC	Report constant (UBE).
RV	Report variable (UBE).
IC	Input column (table conversion).
OC	Output column (table conversion).

BC and FC share the same internal structure if an FC is associated with a database item; filter fields are an exception.

Event Rule Manipulation

To automatically tailor searches to certain situations without adding filter form controls, you can use event rules to assign values to QBE columns. Unlike filters, event rules allow you to change comparison operators at runtime instead of during design time.

Processing Available Objects

When the value of an available object changes during event rule processing, the following actions occur:

- The new value is copied to the business view data, grid data, form control data, form-interconnect data, or processing option data immediately after that ER line is processed (internal runtime structure).
- Form control data and grid data are copied to the appropriate form control or grid cell (external screen).

Form control data that is associated with database items share the runtime data structure with business view data. (Filter fields are an exception to this rule.) This situation means that:

- FC data and BC data are always identical.
- Whenever FC runtime data is changed, any reference to BC reflects the same value.
- Whenever the BC value is changed, the FC runtime value also changes. This change in the FC is not reflected on the form until the FC runtime value moves to the screen.
- On Control is Exited processing, the value entered into the form control is captured in the runtime data structure that is shared by BC and FC.

Control is Exited Processing

Control is Exited processing includes the following:

- The value in the control is saved to internal runtime structures.
- The *Control is Exited* event is processed.
- If the value has changed since the previous time that the control was exited, the following occurs:
 - The system processes the *Control is Exited and Changed - Inline* event.
 - The system launches the asynchronous process (the next three steps are executed on a thread).
 - The system processes the *Control is Exited and Changed - Async* event.
 - The system validates the Data Dictionary.
 - The form control data is formatted and copied back to the control.

For fix/inspect and transaction forms, this logic is also performed for each control on the OK Button.

Grid Processing

You can use system functions to manipulate the selection or sequencing in a grid. These functions give you direct access to the database APIs and the selection and sequencing structures that you use when the runtime engine populates the grid. You can use the following system functions for this purpose:

Set Selection	Creates one element in the select structure.
Set Lower Limit	Creates one element in the select structure. This system function is similar to Set Selection.
Clear Selection	Removes all system-function-defined selection information.
Set Selection Append Flag	Determines whether the system-function-defined information appends to or replaces QBE and filter field information.
Set Sequencing	Creates one element in the sort structure.
c	Removes all system-function-defined sequencing information.

If a headerless detail form contains at least one equal filter or one nonfilter database form control, the system updates the grid records for the form only if they change.

You can also use the *Get Custom Grid Row* event, and system functions, such as *Continue Custom Data Fetch*, for page-at-a-time processing for custom grids. This processing allows you to fetch only a page of data at a time to limit the number of possible records. This is particularly useful for grids in which you might be fetching thousands of records.

Grid columns have type-ahead capability. When a user enters a character in the field, the system searches a history list for a match. If a match exists, it appears in the field with highlighted text. This feature is particularly useful for data entry work because it can reduce the amount of typing required.

Form Flow

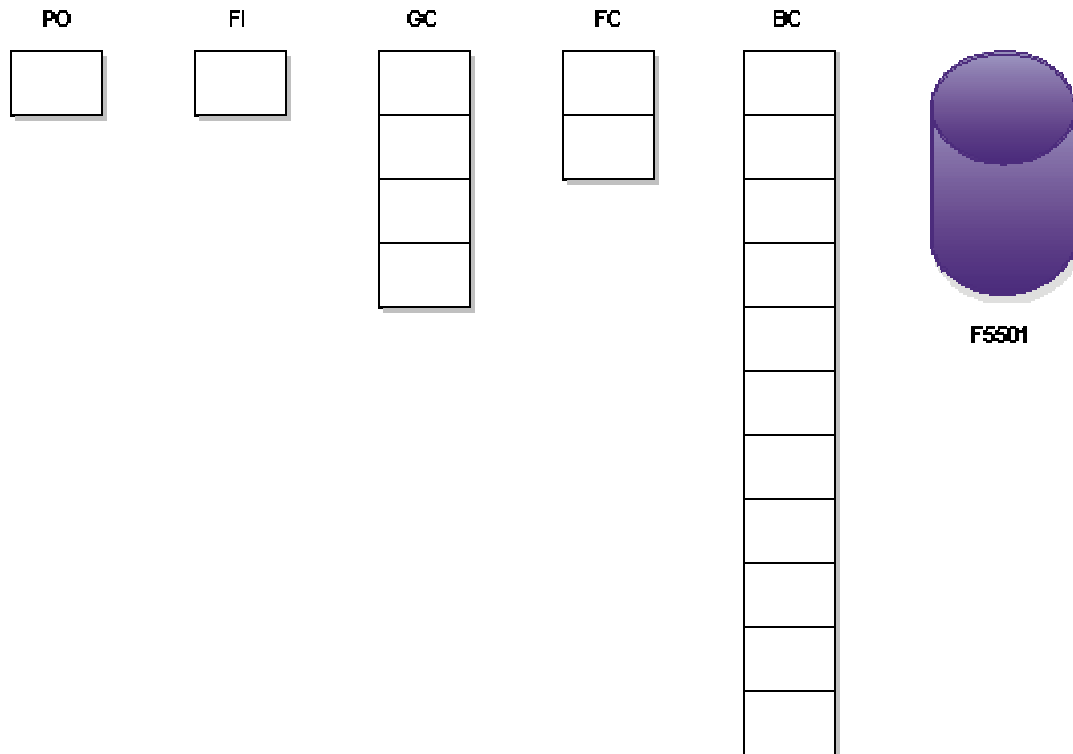
Each form type has its own properties and its own event flow. Processing occurs regardless of whether you add event rule logic. However, the engine pauses at specific events, such as *Dialog Is Initialized*, so that you can add logic or manipulate values. Event rule logic and user interaction determine how the system processes events.

Some of these events occur on each form, although the order and number of events varies depending on the form type. For example, you can use *Dialog Is Initialized* to add logic on all types of forms. Similarly, you can use *Post Dialog is Initialized* to add logic on a fix/inspect form, but not on a find/browse form. For a forms with updateable grids, the events *Dialog is Initialized* and *Post Dialog is Initialized* perform something different than they do for forms without a grid. For forms with an updateable grid, the system performs a Find and reads the database between these two events.

Typical Event Flow for a Find/Browse Form

The following example represents how values in the runtime structures are stored in memory compared to how they appear on the form. This example uses the Find/Browse form when it is called directly from a menu.

Values in Runtime Structures as Stored in Memory



The runtime engine processes events in a certain order. The following topics describe the typical events for the Find/Browse form and the order in which they are processed. This process flow can vary depending on specific user interaction and the event rule logic that you use.

Pre Dialog Is Initialized

The following steps occur before the *Dialog is Initialized* event is processed and the form appears:

- Initialize runtime structures (or clear memory) as follows:
BC = 0

FC = 0

GC = 0

FI = Values passed from a calling form (if any)

PO = Values passed from processing options
- Initialize form controls.
- Initialize error handling.
- Initialize static text.
- Initialize helps.
- Create toolbar.
- Load form interconnect data into corresponding BC columns and filter fields (if any exist).
- Initialize thread handling.
- Use filter controls to build a WHERE clause of an SQL SELECT statement.

If a form interconnection to the form exists, the computer memory contains information in the FI structure. The FI value is copied to the BC runtime structure.

Dialog Is Initialized

The engine pauses for the event to be processed. The system processes all event rule logic that is attached to the Dialog Is Initialized event. When the engine pauses, the runtime structures contain the following values:

BC = Any FI values passed

FC = Any FI values passed

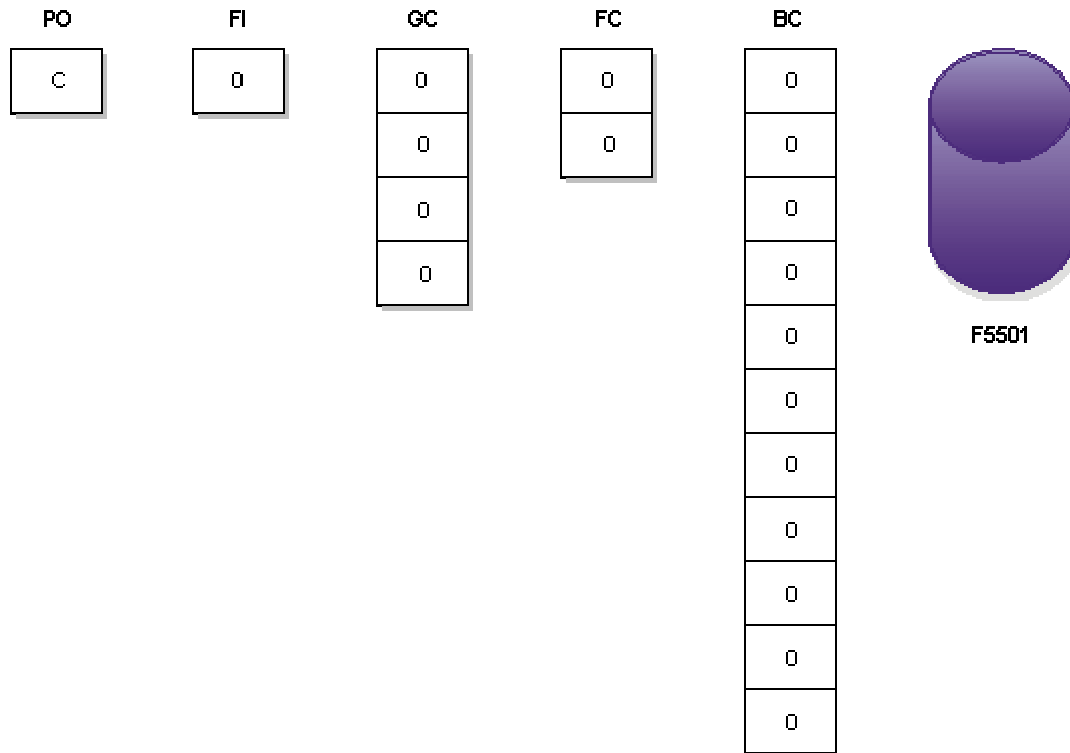
GC = 0

FI = Values passed from a calling form (if any)

PO = Values passed from processing options

The following illustration represents the information in the runtime structures just before the system runs Dialog Is Initialized:

Content of Runtime Structures Before “Dialog Is Initialized” Runs



The Dialog is Initialized event is commonly used to do the following:

- Hide or show controls on a form
- Disable controls

After the system processes the Dialog Is Initialized event, it displays the form.

Post Dialog Is Initialized

The engine pauses for the Post Dialog Is Initialized event to be processed. You can add logic to be processed when the system runs the Post Dialog Is Initialized event. When the engine pauses, the runtime structures contain the following values:

BC = 0 (or values already passed in)

FC = 0 (or values already passed in)

GC = 0 (or values already passed in)

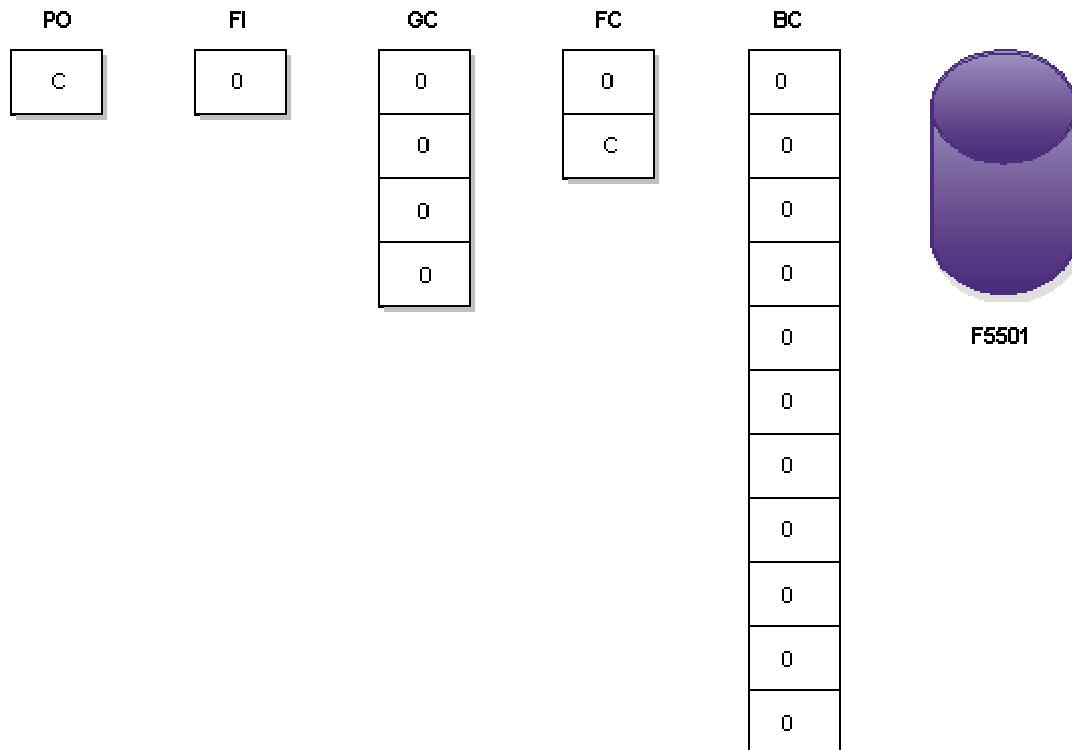
FI = Values passed from a calling form (if any)

PO = Values passed from processing options

At this point, the system has not yet fetched records. To allow you to add logic or manipulate the runtime structure values, the engine pauses just before the Fetch process. You can also perform an Assign process from the filter field here.

The following illustration represents the information in the runtime structures just before the system runs the Form Record is Fetched event:

Content of Runtime Structures Before “Form Record is Fetched” Runs



The Post Dialog is Initialized event is commonly used to do the following:

- Load filter fields that will be used for the `WHERE` clause in the `SQL SELECT` statement
- Load processing option values into filter fields
- Perform any one-time logic for the form, such as fetching a system date

The Post Dialog is Initialized event runs even when a record does not exist.

At this point, the user must click the Find button to populate the grid (the PeopleSoft standard is to disable Autofind), and then the system builds the `SQL SELECT` statement.

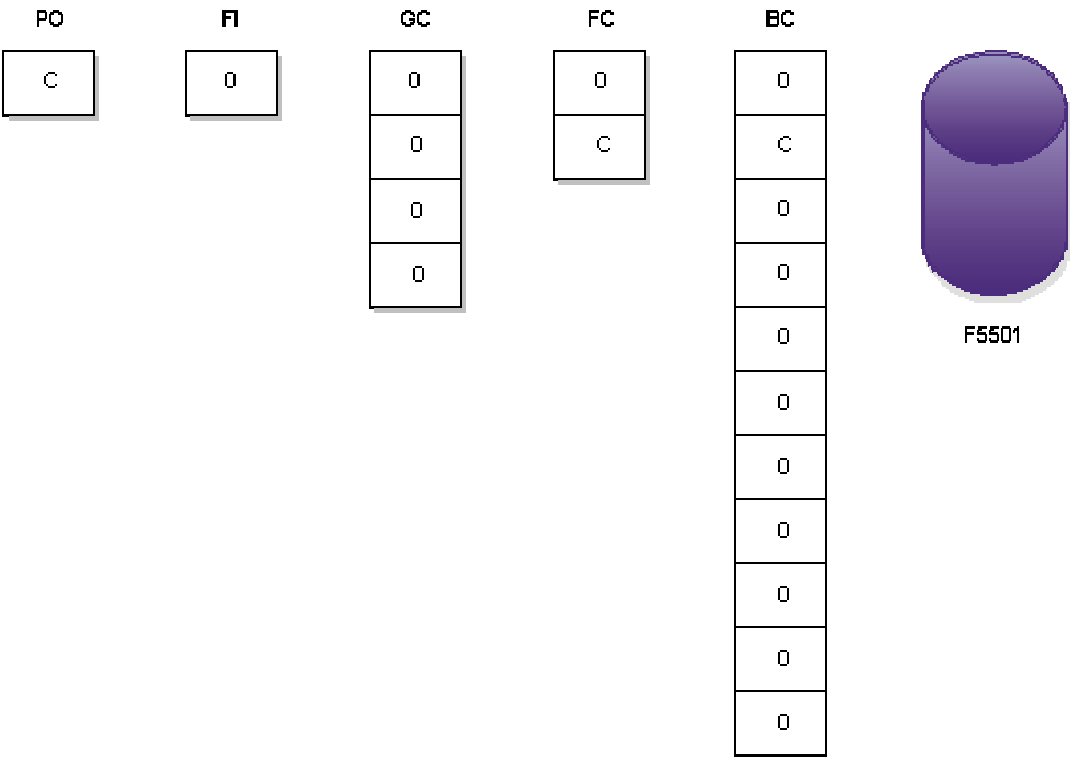
SQL SELECT

After the user clicks the Find button, the system builds a `SELECT` statement with a `WHERE` clause. The `SQL SELECT` statement marks all columns in the business view.

The `WHERE` clause includes any values in the QBE line or in filter fields. The `WHERE` clause is then used to get all records that meet the QBE and filter criteria.

The following illustration represents the information that appears in the runtime structures just before the system builds the `SQL SELECT` and the `FETCH` occurs:

Content of Runtime Structures Before SQL Select Builds



Do in WHILE Loop

Records are fetched one at a time in a `WHILE` loop. The following occurs during the `WHILE` loop:

- The system finds `WHILE` database records, and `WHILE` grid rows are available on the form.
- The system performs page-at-a-time processing.
- `SQL FETCH` gets one record.

Page-at-a-Time Processing

Page-at-a-time processing means that the system fetches only the number of records that can visibly appear in a grid. For example, if the grid displays only three rows, then three records are fetched, one after the other. To fetch the next three, the user must click the down arrow key on the grid scroll bar. You can size the grid when you design the form.

When page-at-a-time processing occurs, the grid display is cached in memory. (This memory cache is different than the runtime structure in memory.) Because the grid is cached, the user can load all records into the grid and then use the up arrow to display a previously fetched record. The cache does not cause a new `FETCH`. For example, assume that records 45 through 47 are loaded and the user clicks the down arrow, after which the system loads records 48 through 50. If the user then presses the up arrow and highlights record 45, a new `FETCH` is not necessary to retrieve record 45 because it is retrieved from the grid cache.

When page-at-a-time processing is turned on, event rule logic attached to the “Last Grid Rec Has Been Fetched” event is not executed unless the user clicks the down arrow on the grid scroll bar enough times to fetch the last record.

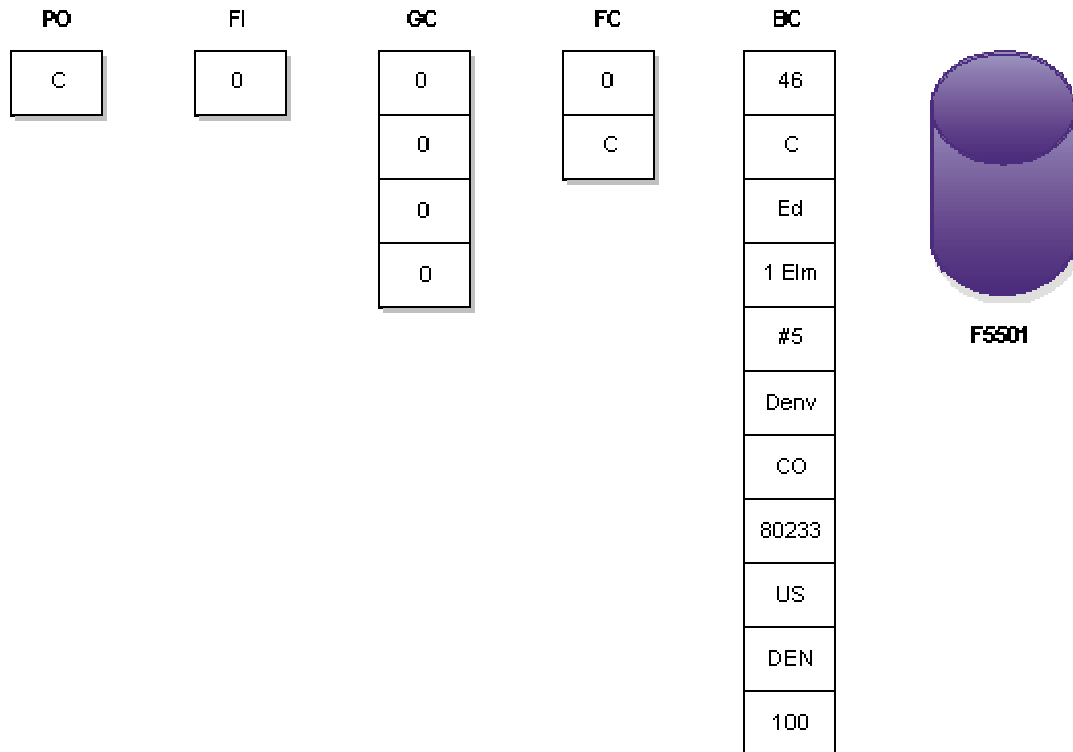
Typically, page-at-a-time processing improves performance. Although it can be disabled, the PeopleSoft EnterpriseOne standard states that you should not disable it unless you have a valid business reason to do so or the form type is Headerless Detail.

BC Assigned Database Values

After the system fetches the first record in the `WHILE` loop, it copies the database values to the BC runtime structure. Values from each marked column in the table appear in the BC runtime structure elements.

The following illustration represents the information in the runtime structures when the system reads the first record in the `WHILE` loop:

Content of Runtime Structures When First Record Read



Grid Rec Is Fetched

The engine pauses again for the event to be processed. You can add logic to be processed when the system runs *Grid Rec Is Fetched*. When the engine pauses, the runtime structures have the following values:

BC = Values from the database (for the first record read)

FC = Values from the database (if the fields are database fields)

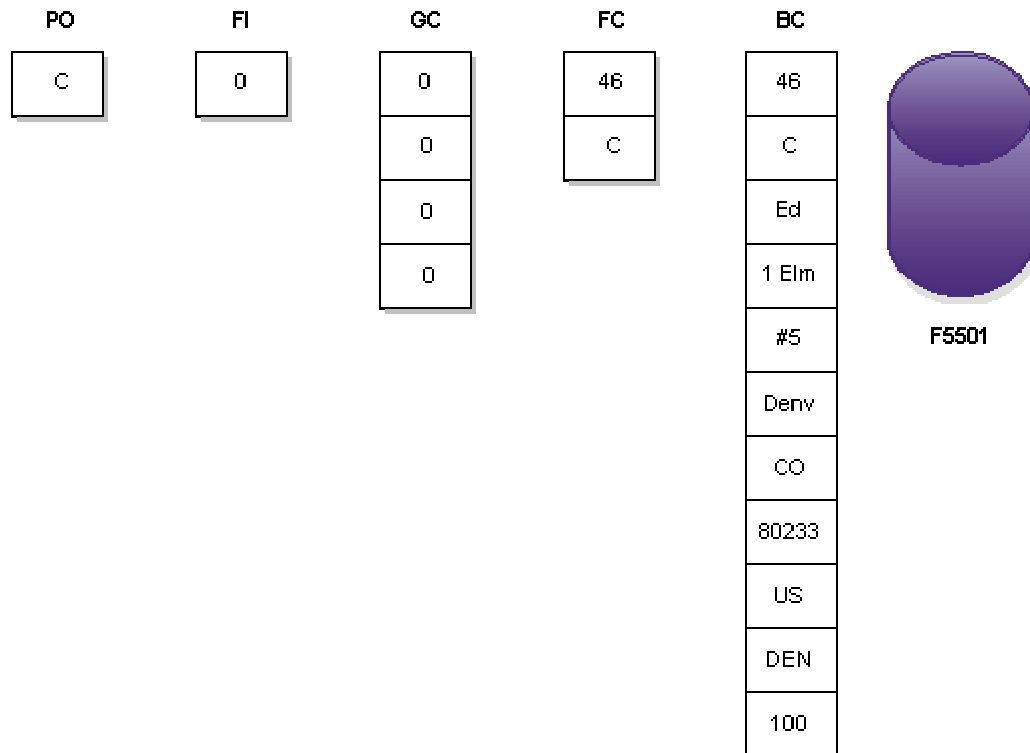
GC = 0

FI = Values passed from a calling form (if any)

PO = Values passed from processing options

The following illustration represents the information in the runtime structures just before the system runs *Grid Rec Is Fetched*:

Content of Runtime Structures Before “Grid Rec is Fetched” Runs



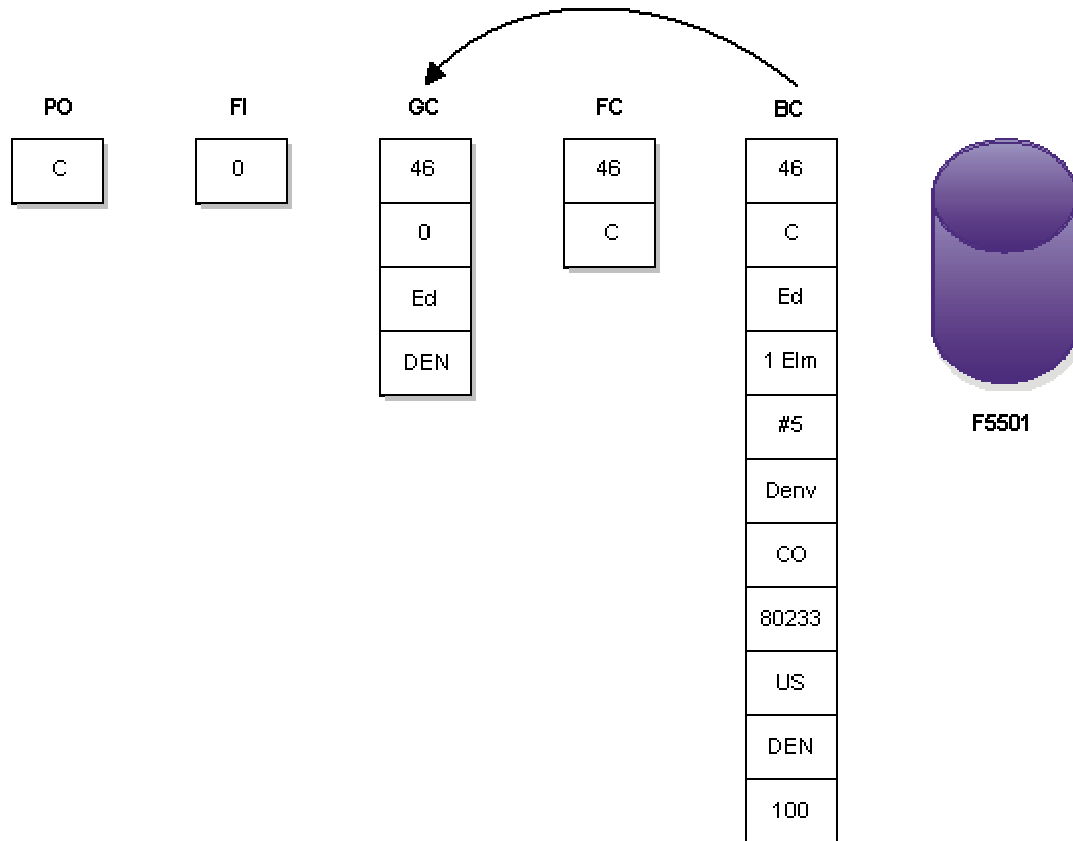
The *Grid Rec Is Fetched* event is commonly used to perform the following actions:

- Calculate a value for a work field in the grid.
- Suppress a row from being written to the grid.

After the *Grid Rec Is Fetched* event (when the first record in the WHILE loop is fetched), the BC values are copied into the GC runtime structure.

The following illustration represents the information in the runtime structures when the system reads the first record in a `WHILE` loop.

Content of Runtime Structures When First Record is Read



Write Grid Line Before

The engine pauses again for the event to be processed. You can add logic to be processed when the system runs *Write Grid Line Before*. When the engine pauses, the runtime structures have the following values:

BC = Values from the database (from the record just read)

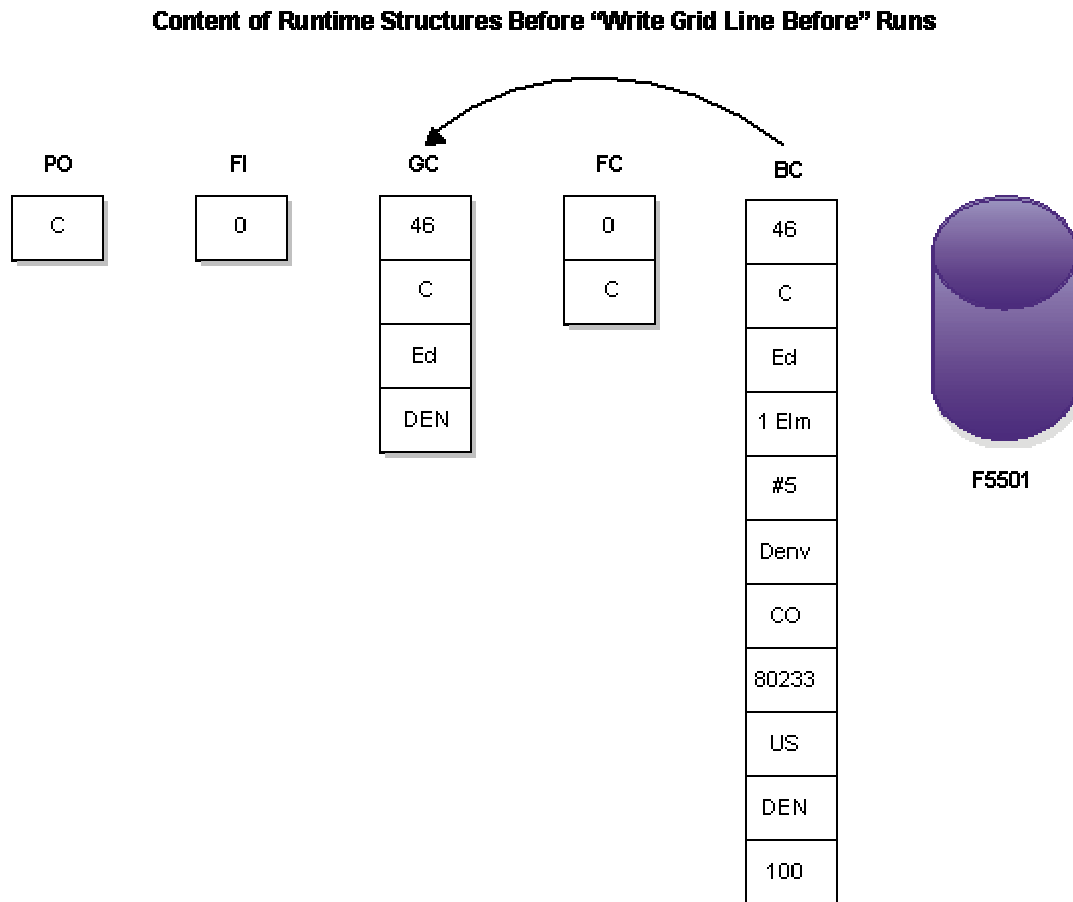
FC = Values from the database (if the fields are database fields)

GC = Values from the database (from the previous read)

FI = Values passed from a calling form (if any)

PO = Values passed from processing options

The following illustration represents the information in the runtime structures just before the system runs *Write Grid Line Before*:



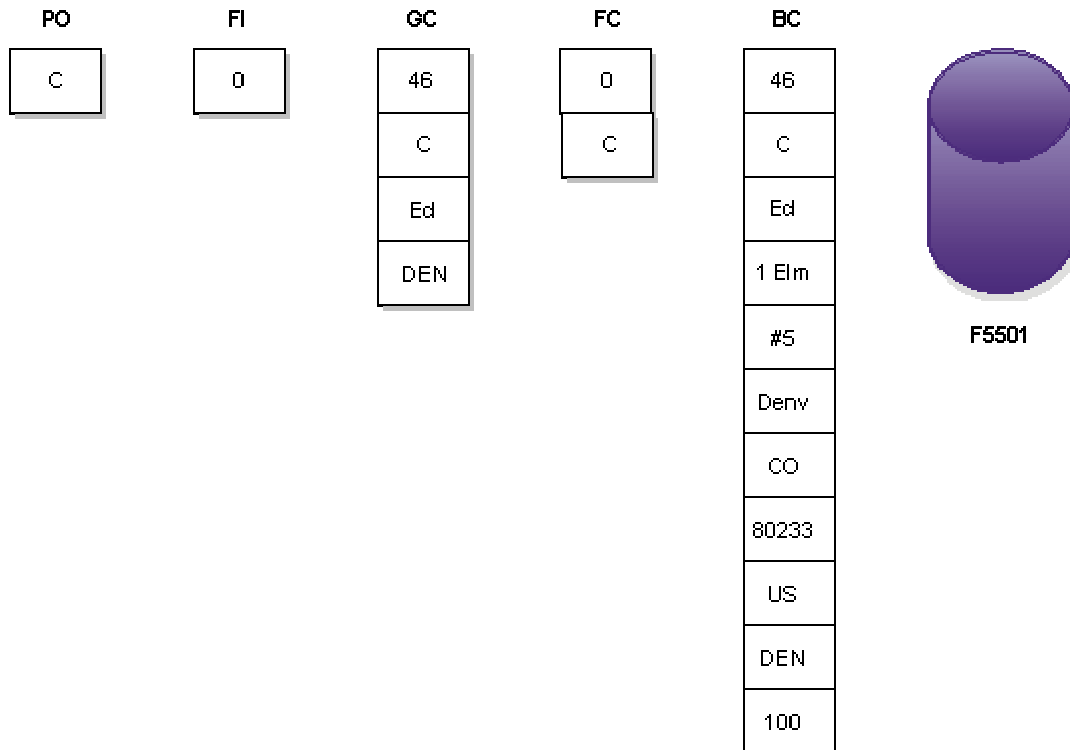
The *Write Grid Line Before* event is commonly used to do the following:

- Suppress a grid row from being written.
- Add logic before the user sees a row on the form.
- Change formatting of a grid column.
- Convert any grid value, such as unit of measure.
- Retrieve additional information for the grid row, such as a description, from tables that are not in the business view.

After the system processes *Write Grid Line Before*, the GC elements, which now include the database values for the first record, are copied to the grid cells on the form.

The following illustration represents the information that appears in the runtime structures now:

Content of Runtime Structures After “Write Grid Line Before” Runs



Write Grid Line After

The engine pauses again for the event to be processed. You can add logic to be processed when the system runs *Write Grid Line After*. When the engine pauses, the runtime structures have the following values:

BC = Values from the database (from the first record read)

FC = Values from database (if the field is a database field)

GC = Values from the database (from the first record read)

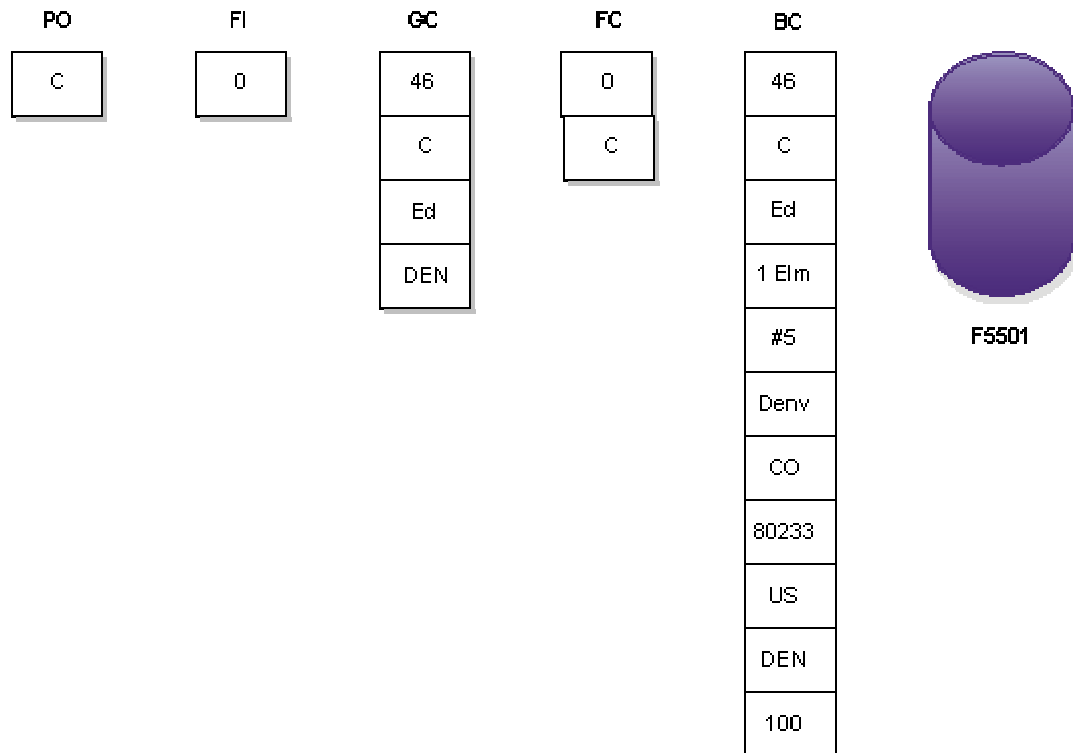
FI = Values passed from a calling form (if any)

PO = Values passed from processing options

The system displays the current record in the grid cells.

The following illustration represents the information in the runtime structures just before the system runs *Write Grid Line After*:

Content of Runtime Structures Before "Write Grid Line After" Runs



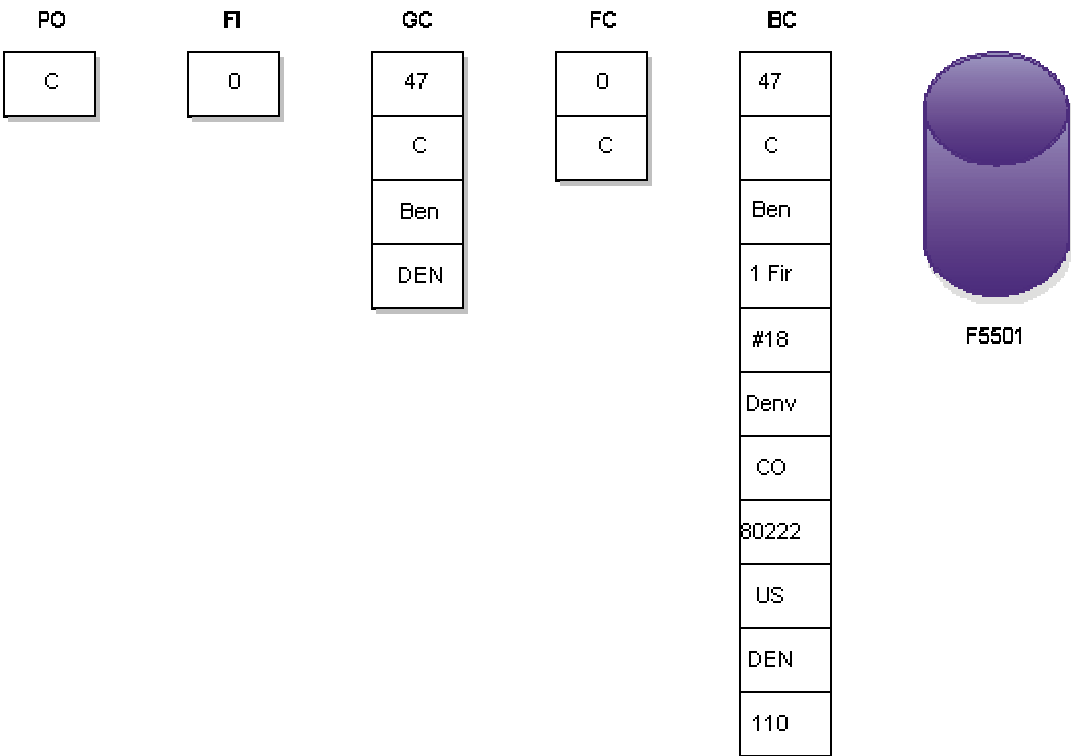
You typically use the *Write Grid Line After* event to add logic after the user sees a row on the form.

The system reads records in the database one at a time and performs the same processing steps. When the system reads the next record, it performs the following processing steps:

- Performs an SQL Fetch of next record that matches criteria
- Assigns BC values from the database
- Processes *Grid Rec is Fetched*
- Assigns BC values to GC
- Processes *Write Grid Line Before*
- Displays values in the grid row on the form
- Processes *Write Grid Line After*

The following illustration represents the information in the runtime structures after the system processes the *Write Grid Line After* event for the second record:

Content of Runtime Structures After “Write Grid Line After” Runs



If the system locates grid records and the user clicks the arrow keys to fetch all records that meet the selection criteria, the SQL `FETCH` finally fails. If the grid is full and the user never clicks the down arrow key, `FETCH` might not fail because the system might not have read some of the matching records. When `FETCH` fails, the system processes *Last Grid Rec Has Been Read*.

Last Grid Rec Has Been Read

The engine pauses again for the event to be processed. You can add logic to be processed when the system runs *Last Grid Rec Has Been Read*. When the engine pauses, the runtime structures contain the following values:

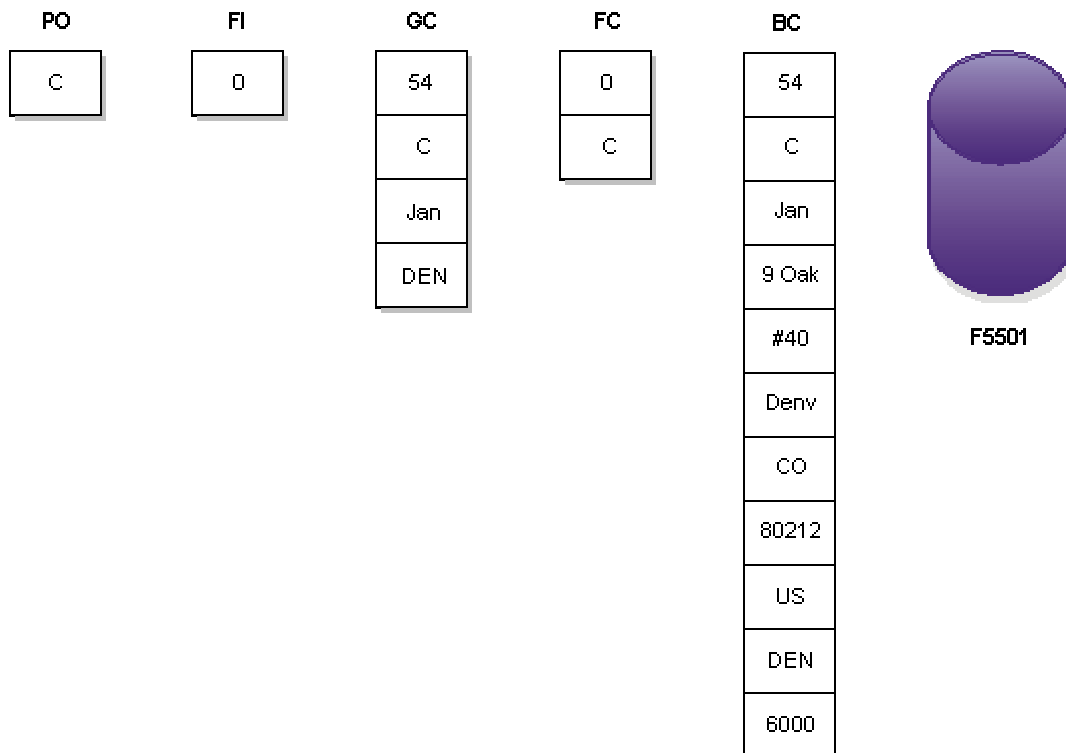
- BC = Values from the database (from the last record read)
- FC = Values from the database (if the field is a database field)
- GC = Values from the database (from the last record read)
- FI = Values passed from a calling form (if any)

PO = Values passed from processing options

The GC values appear on the form.

The following illustration represents the information in the runtime structures just before the system runs *Last Grid Rec Has Been Read*:

Content of Runtime Structures Before “Last Grid Rec Has Been Read” Runs



The *Last Grid Rec Has Been Read* event is commonly used to do the following:

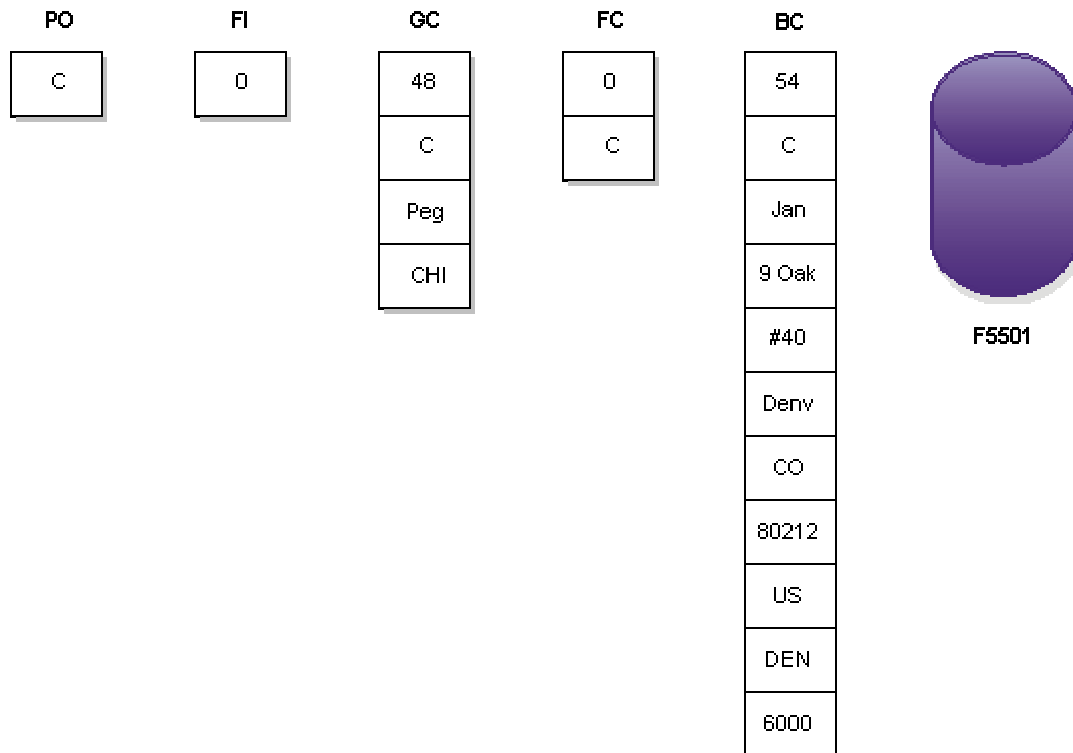
- Write total lines to the grid.
- Display totals that are based on grid values.

Select Button Processing

When a user chooses a grid row and clicks the Select button, the system copies form values back into the GC structure. The BC structure stays the same, and the system does not update the database just because the user clicked a row. Therefore, you should select GC as the value to pass during an interconnection.

The following illustration represents the information in the runtime structures when the user chooses a grid row. Note that the BC and GC structures do not contain the same values.

Content of Runtime Structures When User Chooses Grid Row



Button Clicked

The engine pauses for the event to be processed. You can add logic to be processed when the system runs *Button Clicked*. When the engine pauses, the runtime structures have the following values:

BC = Values from the database (from the last record read)

FC = Values from the database (if the field is a database field)

GC = Values from the selected grid row

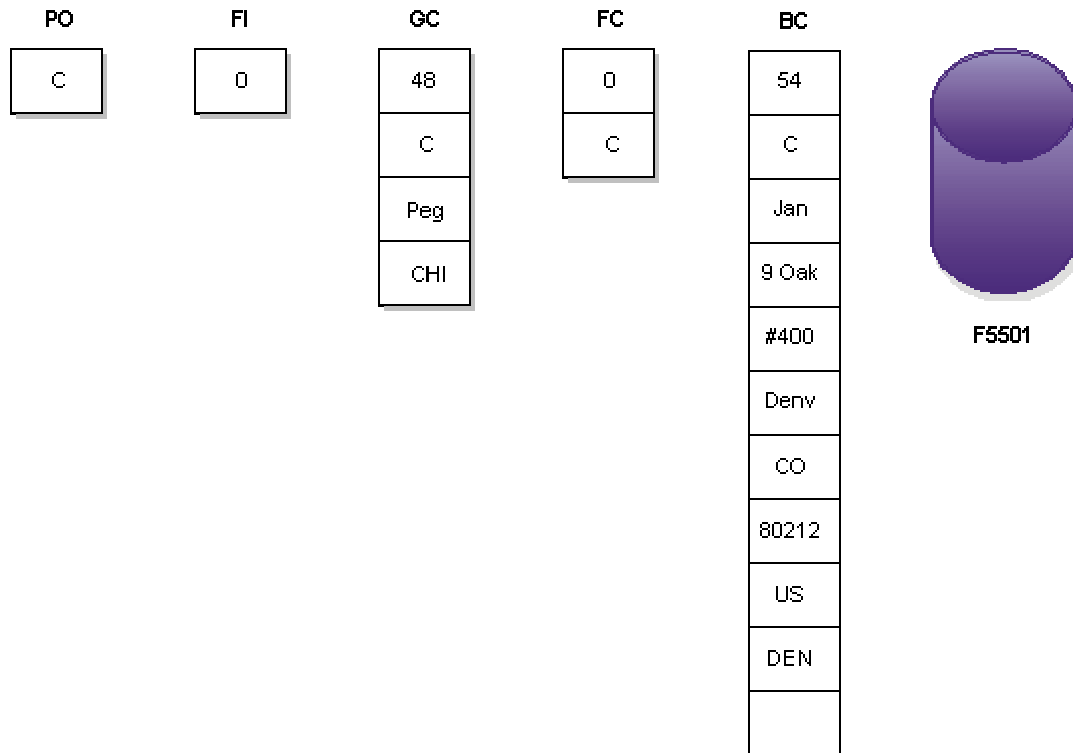
FI = Values passed from a calling form (if any)

PO = Values passed from processing options

The form grid cells display the current record.

The following illustration represents the information in the runtime structures just before the system runs *Button Clicked*:

Content of Runtime Structures Before “Button Clicked” Runs



The *Select Button Clicked* event is commonly used to do the following:

- Connect to another form.
- Pass GC values from the form into the FI structure of a receiving form.
- Use *Repeat Business Rules for Grid* to repeat event rules when multiple rows are selected (you must also enable the form property *Multiple Select* for the multiselect feature to work).

Note

Activate *Repeat Business Rules for Grid* if you want to allow the user to choose multiple grid rows.

Add Button Processing

After the user chooses a grid row, the GC runtime structure is assigned the values that appear in the grid record on the form. Normally, the user does not choose a row before an add action, but if a row is highlighted, the system copies form values to the GC runtime structure. If the user does not choose a record, the GC runtime structure contains the values from the last grid record that the system read, if any. The system does not update the database just because the user clicks the row.

The engine pauses for the *Button Clicked* event to be processed. You can add logic to be processed when the system runs *Button Clicked*. When the engine pauses, the runtime structures have the following values:

BC = Values from the database (from the last record read)

FC = Values from the database (if the field is a database field)

GC = Values from the selected grid row

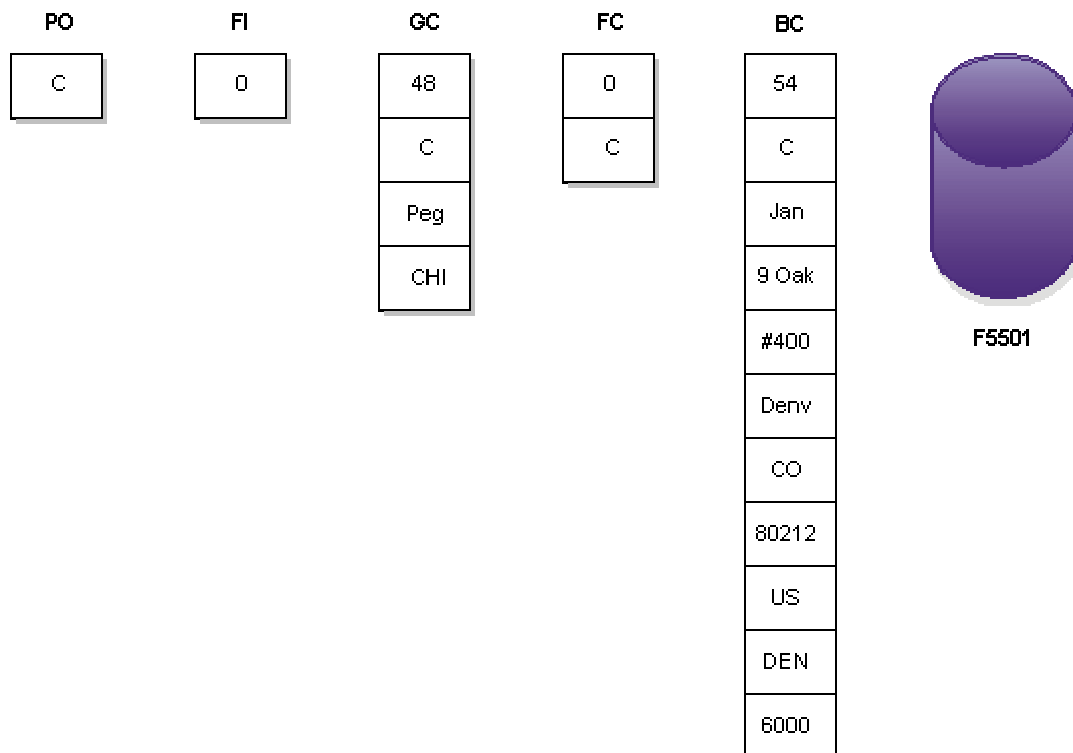
FI = Values passed from a calling form (if any)

PO = Values passed from processing options

Because this is an add action, the content of GC is irrelevant at this point. BC and GC do not contain the same values.

The following illustration represents the information that is in the runtime structures just before the system runs *Add Button Clicked*:

Content of Runtime Structures Before “Add Button Clicked” Runs



You typically use the *Add Button Clicked* event to interconnect to another form, such as a fix/inspect or headerless detail form on which the system actually performs the add action.

Because you are adding a new record, you generally do not pass GC values to the FI structure of the receiving form here.

Delete Button Processing

When the user chooses a grid row, the GC runtime structure is assigned the values that appear in the grid record on the form. BC is still the same, and the system does not update the database.

The engine pauses for the *Button Clicked* event to be processed. You can add logic to be processed when the system runs *Button Clicked*. When the engine pauses, the runtime structures have the following values:

BC = Values from the database (from the last record read)

FC = Values from the database (if the field is a database field)

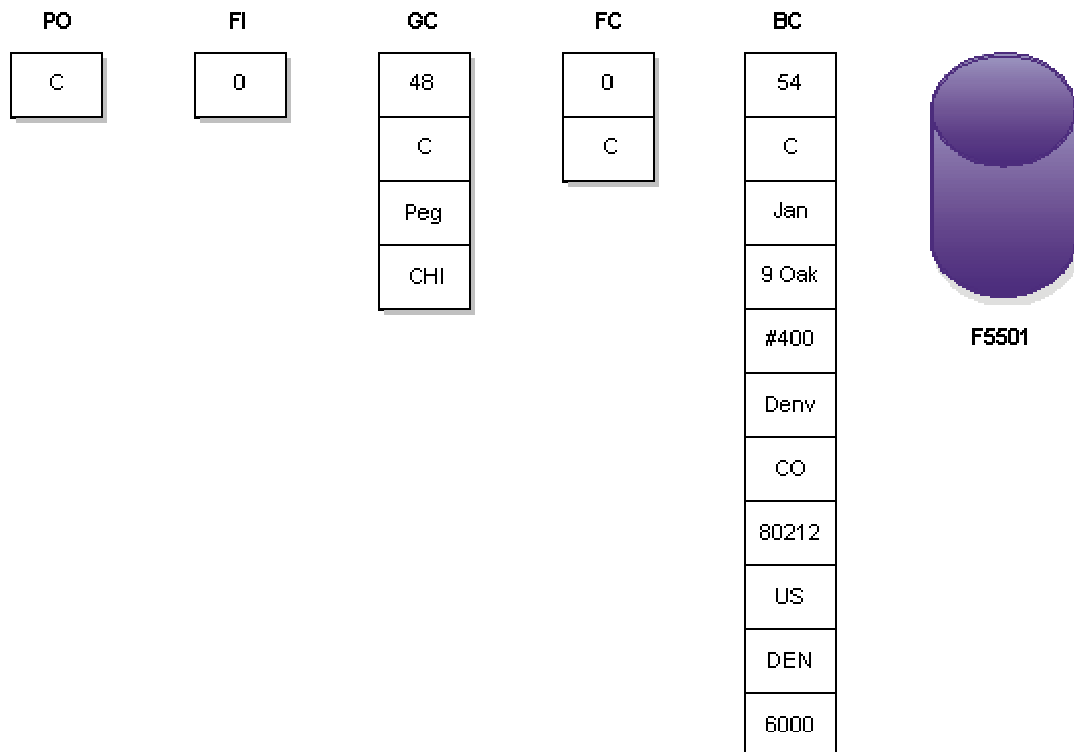
GC = Values from the selected grid row

FI = Values passed from a calling form (if any)

PO = Values passed from processing options

The following illustration represents the information in the runtime structures just before the system runs *Button Clicked*:

Content of Runtime Structures Before Delete "Button Clicked" Runs

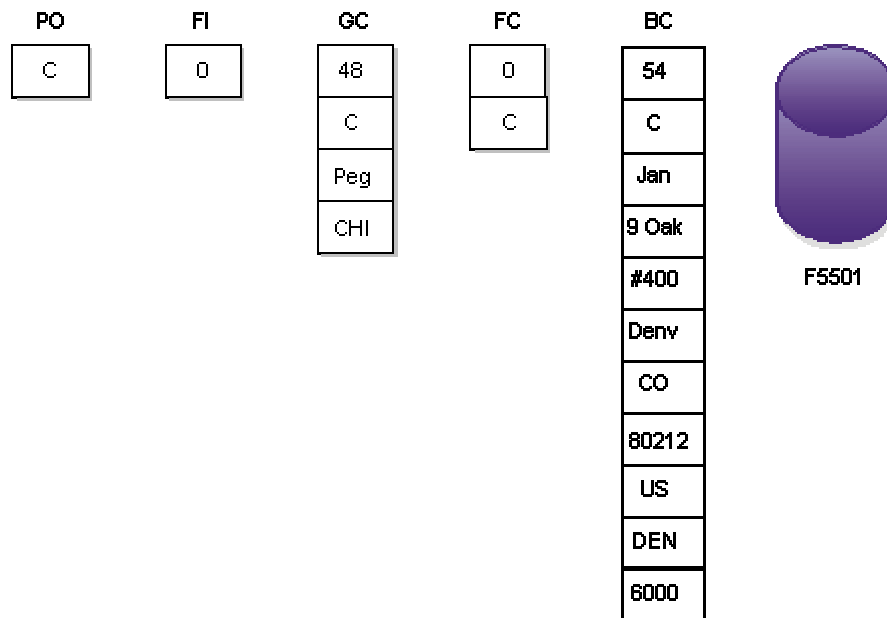


Next, the *Delete Grid Rec Verify Before* event occurs.

Delete Grid Rec Verify Before

The engine pauses for the *Delete Grid Rec Verify Before* event to be processed. You can add logic to be processed when the system runs *Delete Grid Rec Verify Before*. The system processes the logic that is attached to this event after the user clicks Delete, but before the pop-up Verify confirmation form appears.

Content of Runtime Structure When “Delete Grid Rec Verify Before” Runs



Next, the *Delete Grid Rec Verify After* event occurs.

Delete Grid Rec Verify After

The engine pauses for the *Delete Grid Rec Verify After* event to be processed. You can add logic to be processed when the system runs *Delete Grid Rec Verify After*.

You might want to perform editing to verify that the delete is valid. For example, other tables might contain dependant records that prevent this record from being deleted as long as they exist.

The system processes the logic that is attached to this event after the user clicks OK in the Verify confirmation form. If the user clicks Cancel in the Verify confirmation form, the logic attached to this event does not occur.

Next, the *Delete Grid Rec From DB Before* event occurs.

Delete Grid Rec From DB Before

The engine pauses again for the *Delete Grid Rec From DB Before* event to be processed. You can add logic to be processed when the system runs *Delete Grid Rec From DB Before*. When the engine pauses, the runtime structure FC is blank.

The system processes the logic that is attached to the *Delete Grid Rec From DB Before* event after the user clicks OK to delete a record and after it processes the Verify window, but before it actually deletes the record.

You can use the *Suppress Delete* system function with the *Delete Grid Rec From DB Before* event to turn off the automatic tool delete and perform the delete yourself. For example, you can use a business function to perform the delete.

After the system processes the *Delete Grid Rec From DB Before* event, it builds an SQL DELETE statement. The engine pauses again so that you can add logic to process. FC is blank. The SQL DELETE occurs at this point, and the system deletes the current record from the database. When you choose multiple records, one delete call can delete all of the records.

Delete Grid Rec From DB After

The engine pauses again for the *Delete Grid Rec From DB After* event to be processed. You can add logic to be processed when the system runs the *Delete Grid Rec From DB After*. This logic runs after the system physically deletes the record from the database. You might use this event to call a business function to delete information from related tables that are not in the business view.

The system processes the logic that is attached to the *Delete Grid Rec From DB After* event after the user clicks OK to delete the record and after the system processes the Verify confirmation form and deletes the record.

All Grid Recs Deleted From DB

The engine pauses again for the *All Grid Recs Deleted from DB* event to be processed. You can add logic to be processed when the system runs *All Grid Recs Deleted from DB*. At this point, FC is blank.

You can also add logic after the system physically deletes the record from the database. The system processes logic that is attached to the *All Grid Recs Deleted from DB* event after it deletes multiple grid lines and the corresponding database records. Rules that are attached to this event are not apparent on the form and cannot manipulate the grid lines or records.

Working with Event Rules Design

You can use Event Rules Design to create event rule logic for controls on a form. For example, assume that you want to pass data for a selected record on a find/browse form to a fix/inspect form to revise that record. To accomplish this task, you would create a form interconnect event rule and attach it to the Select toolbar option for the Button Clicked event.

Before you create an event rule, consider the control (form, button, field, grid, and so on) and the event upon which the event rule should process. Answering the following questions will determine upon which of the events logic should occur:

- Is the user initializing the form?
- Is the user clicking a button?

- Is the user exiting from a field?
- Is the user changing or exiting from a row?

Creating and Saving Event Rules

After you place controls on a form, you must create event rule logic to cause processing to occur within your application at runtime. Remember that a form is also a control, and you can create logic that the system automatically processes whenever a form is initialized.

You create event rules by clicking the buttons on the toolbar in Event Rules Design. Depending on the button that you click, a different work area appears for creating and manipulating the event rule line-by-line. Specific buttons within Event Rules Design allow you to:

- Attach a business function.
- Attach a system function.
- Create an If/While statement.
- Assign a value or expression.
- Create a form interconnection.
- Create a report interconnection.
- Insert an Else clause in an If statement.
- Create a variable.
- Perform table I/O.

► To create and save an event rule

1. On Form Design, choose a control on the form.
2. From the Edit menu, choose Event Rules.
3. Choose an event from the Events list box (for example, *Control Is Entered*).
4. Click one of the following Event Rules buttons to add logic for the control:
 - Assignment/Expression
 - If/While
 - Business Function
 - System Function
 - Method Invocation
 - Variables
 - Else
 - Table I/O
 - Report Interconnect (UBE)
 - Form Interconnect

5. Click the following buttons to add a note, enable or disable a specific line within the event rule, or delete a specific line within the event rule:

- Comment
- Enable/Disable
- Delete

The Enable/Disable button affects only a specific line within the event rule. To disable the entire event, select the Disable Event option from the Options menu.

You can disable a specific line within an event rule when you want to test your event rule logic. When you are confident that a line is no longer needed, you can delete it from the event rule.

To enable, disable, or delete multiple consecutive lines, press the shift key and choose all event rules that you want to enable, disable, or delete. After you have chosen the desired lines, click the Enable/Disable or Delete button.

6. Click Save to save the event rule.
7. Click OK to return to Form Design.

Saving the event rule saves it with the application. If you delete the application, or if you cancel the application before you save it, the event rule is deleted also.

Locating Event Rules

If the logic for an event rule becomes unwieldy, you can use the Find feature to locate specific strings within the event rule.

► To find a string of text in an event rule

1. From Event Rules Design, choose Find from the Edit menu.
2. Click the appropriate option to Indicate the direction in which you want to search.
3. Enter the string of text that you want to find and click the Find Next button.

Cutting, Copying, and Pasting Event Rules

You can cut or copy event rules and paste them in the same event, form, or application or in a different event, form, or application.

You can also paste event rules into other applications, such as word processing documents. This feature is useful for documenting your project.

You can use any of the following buttons:

- Cut
- Copy
- Paste
- Paste Options

When you cut event rules, the system deletes the selected lines from the source and stores them on the clipboard.

► **To cut event rules**

1. On Event Rules Design, choose the lines of event rules that you want to cut.
2. Click the Cut button.

► **To copy event rules**

1. On Event Rules Design, choose lines of event rules that you want to copy.
2. Click the Copy button.

When you copy event rules, the system copies the selected lines of event rules and stores them on the clipboard.

► **To paste event rules**

When you paste event rules, the system resolves objects from the source as you past them. If an object is partially resolved, the system pastes the closest matching object from the destination event rules. A comment line appears above the partially-resolved line of event rules and in the status bar to indicate that the object is partially resolved.

1. On Event Rules Design, choose the line after which you want to insert your lines of event rules.
2. Click the Paste button.

Some objects cannot be resolved in the destination event rules. The system disables these lines of event rules and displays a comment. For example, an EndIf statement is commented out if its associated If statement is missing.

For criteria statements, the paste operation adds whatever is necessary to maintain a clean, logical structure. For example, if you paste an If statement and no EndIf statement exists, the paste operation adds a matching EndIf statement to make the logic complete.

You can set paste options to display comments before and after a block of pasted event rules.

► **To set paste options**

1. On Event Rules Design, choose Paste Options from the Edit Menu.
2. On Cut Copy Paste Options, click the Paste Options option.
3. Enter the number of lines for which you want comments to appear.

When you paste information, comments indicate where your pasted selection begins and ends.

Adding Comment Lines to Event Rules

You can add a comment line to document event rule code.

► To add a comment line to an event rule

1. On Event Rules Design, position the cursor where you want to add a comment.
2. Click the Comment button.
3. Type a comment and then click OK.

The system automatically formats the comment in Visual C++ notation so that you do not have to type the symbols (`//`) to indicate that the inserted text is a comment.

Printing Event Rule Logic

You can print the code for event rules. This is especially useful when many lines of code make up an event rule. You can print event rule code for any of the following:

- An entire application
- A form
- A control
- A single event

► To print event rule code

1. On Form Design Aid, click the control for which you want to print event rule code.
2. From the Edit menu, choose Event Rules.
3. On Event Rules Design, choose an event from the Events list box.

The event rule appears in the work area.

4. From the File menu, choose Print.
5. On Event Rules Printing, click one of the following destinations:
 - Printer
 - File
6. Choose one of the following print scopes:
 - Application
 - Form
 - Control
 - Event
7. Choose one of the following page break options:
 - Application
 - Form

- Control
 - Event
8. Choose one or more of the following print ER options:
- Expand Arguments
 - Date/Time Stamp
 - Show Line Numbers
 - Show Comments
9. Click OK.

Validating Event Rules

You validate event rules to help you locate errors, such as a variable or business function that no longer exists or an incorrect business view column. You can validate event rules as you are working or when you save an application.

The system validates event rules either when you click Save or when you choose Exit from the File menu and save before exiting. You can exit Form Design Aid or leave it open if the system locates errors during validation upon exit.

You can also choose the menu option entitled Validate Event Rules. When you do so, the system immediately validates event rules. Validate Event Rules checks for errors at two levels. The first level occurs in Business Function event rules or Table event rules, in which the system converts the event rule code from the event rule scripting language to C code and then to compiled code. The second level occurs in the application, in which the system reviews the event rules for errors such as data type mismatches, data structure errors, and missing variables and controls. The Generator does not check logic errors or syntax.

The error log that the system creates is stored in a file, such as `b9\prod\log\p1234.log` (where *prod* is your environment). If no errors exist, the system does not generate a log.

► To validate event rules

On Form Design Aid, from the File Menu, choose Validate Event Rules.

The Validate Event Rules form displays messages as it processes. When the validation successfully completes, a Validation Successful message appears.

Working with If and While Statements

If and While statements are conditional instructions for an event rule. They evaluate conditions and dictate the flow of logic when the event rule is activated.

If Statements

An If statement executes conditional event rule logic. That is, the logic executes only *if* the condition is true. The system evaluates an If statement only once when it processes the event rule.

If statements are typically formatted as follows:

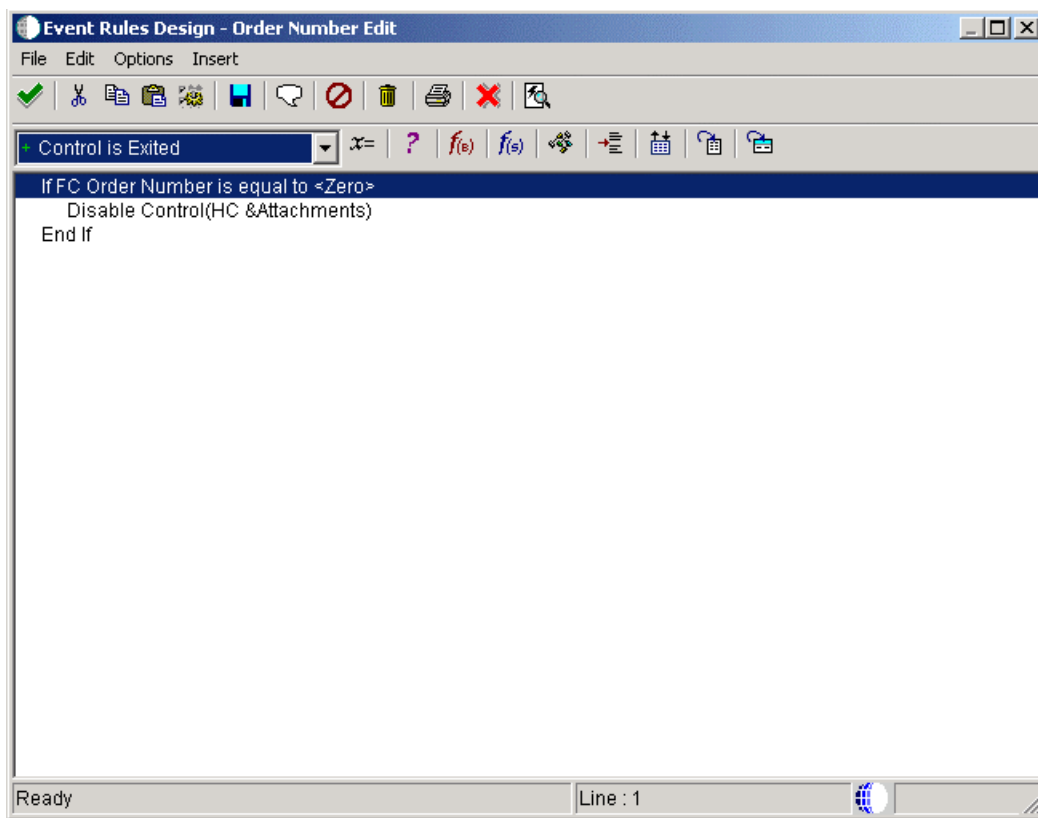
```
IF condition X is true  
    Do Y  
ELSE  
    Do Z  
ENDIF
```

The system evaluates an If statement as follows:

If condition X is true, then do Y and do not evaluate the Else statement. If condition X is not true, then proceed to the Else clause and do Z.

The Else clause is optional. When no Else clause is necessary, the system proceeds to the next statement and evaluates it.

The following is an example of an If statement form Sales Order Entry:



While Statements

A While statement repeats conditional event rule logic. That is, the logic continuously executes as long as a condition is true. A While statement controls the execution of a *loop*, or repetitive test.

A While statement is typically formatted as follows:

```
WHILE condition X is true  
    Do Y  
ENDWHILE
```

The system evaluates a While statement in the following manner:

Evaluate condition X. If it is true, then do Y. Repeat evaluation of condition X until it becomes false, and then exit the loop.

Creating If and While Statements

You use the If/While button to build conditional logic into an event. When you create an If statement, the system also automatically inserts an Else clause. However, you can use the Delete button to delete the Else clause and then reinsert it using the Insert Else button. When you delete an If or While statement, the system also deletes the associated Else and Endif or Endwhile clauses, but not the rules inside of those statements.

You can drag and drop statements line-by-line to change their sequence. Resequencing event rules can result in improper syntax. When you click the Save button or the OK button, the system verifies the syntax. If it detects syntax errors, you can either disable the event rule and continue or edit the event rule to eliminate the errors.

The only valid logical operators when you use range and list values are Is Equal To and Is Not Equal To. The criteria design tool prevents you from using the range and list literals incorrectly by ensuring that the comparison is either equal to or not equal to. If you choose Range or List and your comparison field does not contain an Is Equal To or Is Not Equal To comparison, then the cell is cleared.

To change a statement, click the cell to select the cell or value that you want to change. You can then change the value of that cell. The system checks the syntax when you click OK. To change the sequence of a line, choose the entire row by clicking the row header. Then use the up and down arrows on the toolbar to move that line. You cannot move the If/While line.

To delete a statement, choose the line of the statement, and then click Delete. You cannot delete the top-most clause of the statement (If or While) in If/While design; you can only delete it from the Event Rules Design window.

► To create an If or a While statement

1. On Event Rules Design, choose an event in the Event Rules Design window and click the If/While button.

Each cell in the Criteria Design grid represents a component of the criteria. When you choose a cell by clicking on it or using the Tab key to move into it, a list of valid options appears. To choose an option, either double-click it or select it, and press Enter or Tab. You can choose an option by using the mouse or by typing the option that you want.

Criteria Design has a type-ahead feature that allows you to type the first few characters of an item in a field to automatically display a list of available items that begin with those letters.

2. Choose one of the following operators:

- If
- While

3. Choose a left operand from the list of available data items.

You can right-click to sort the available data items by name or object type. If only one type exists, the sort options are not available.

You can group the available data items by the following object types:

BC A column in the business view for this form

GC A column in the grid for this form

FI A value passed through form interconnection to this form

FC A control on this form, such as a push button

PO A value from processing options of the application

HC A hypercontrol

SV A system variable

SL A system value

VA A variable

4. Choose a comparisons from the following list of logical operators:

- is equal to
- is not equal to
- is less than
- is less than or equal to
- is greater than
- is greater than or equal to

5. Choose a right operand from the object list.

6. To assign a literal, select <Literal>.

The Literal dialog depends on the data item.

7. Click Save to save the criterion statement and return to the Event Rules Design window.

If the criteria are incomplete (for example, if the right operand is missing), you must fix the criteria or delete it.

To create complex If statements, you can select the And option or the Or option, and continue your logic.

Defining Literal Values in Event Rule Logic

You can define literal values in event rules.

► To define literal values

1. On Criteria Design, from the Right operand list, double-click <Literal>.
2. Click one of the following tabs:
 - Single value
 - Range of values
 - List of values
3. Complete the appropriate fields on the tab that you chose.
4. Click OK.

Working with Event Rule Variables

An event rule variable is associated with a data dictionary item, but it does not reside in the data dictionary. After you create it, you can use it only in a specific interactive application, batch application, or business function event rule.

The system automatically initializes the event rule variable at runtime because you define how the event rule variable is used when you create it.

Use event rule variables instead of hidden fields. Event rule variables use fewer system resources at runtime.

The scope of an event rule variable determines how you can use it. Different scope options are available for interactive and batch applications. For example, you can do the following:

- Reference a report variable anywhere in the report.
- Reference an event variable only within the event in which it was created.

Interactive Event Rule Variables

Interactive event rule variables are available at the following levels:

- Form level
- Grid level
- Event level

Form Level

Form-level variables are available at all events for all controls within the form. They are initialized when the form is initialized, and they retain values until the form is closed.

Grid Level

Grid-level variables are subtypes of form variables, and they are available on any form that includes a grid. They are available from all events on the form. They apply to the current row. Every new row added to the grid has the same set of variables. These variables are re-initialized each time a new row is added to the grid. You can use these variables as temporary work fields for the grid. For better system performance, you should use variables instead of hidden work fields whenever possible. Using variables instead of hidden work fields enhances the performance of PeopleSoft EnterpriseOne forms during initialization and saves time because variables are not formatted. Although grid variables are available in all event-rule line types, you should use them only with events that are associated with grid rows.

Event Level

An event-level variable is available only within the event for the form control for which it was added. The variable is re-initialized each time the event is processed for the form control.

Batch Application Event Rule Variables

Batch event rule variables are available at the following levels:

- Report
- Section
- Event

Creating an Event Rule Variable

After you create an event rule variable, it appears in the available objects list in Event Rules Design where you added it. Use the event rule variable in event rules just as you would use any available object in the list. If you create an event level variable and do not use it in event rules, Form Design Aid automatically deletes it.

After you add an event rule variable, you cannot be modify it. Instead, you must delete it and create another one.

The system automatically assigns to each variable one of the following prefixes, based on the specified scope:

- frm_ (Form)
- evt_ (Event)
- grd_ (Grid)
- rpt_ (Report)
- sec_ (Section)

► **To create an event rule variable**

1. On Event Rules Design, click the Variables button.

The Variables form displays different scope options, depending on whether you are working with an interactive application, batch application, or business function event rule.

2. Complete the variable naming field located under the Add button.

Note

Event Rule variables should be formatted as follows:

xxx_yyzzzzzzz_AAAA

Where:

xxx = A prefix that PeopleSoft EnterpriseOne software automatically assigns, depending on the scope. Examples include:

frm_ (form scope)

evt_ (event scope)

yy = Hungarian Notation for the following C variables:

c = Character

h = Handle request

mn - Math numeric

sz - String

jd - Julian date

id - Pointer

zzzzzz = A variable name that you assign; capitalize each word

AAAA = A data dictionary alias (all upper case)

Do not include any spaces in the event rule variable. For example, a branch/plant event rule variable would be:

evt_szBranchPlant_MCU

3. Click one of the following Scope options, depending on the purpose for which you created the variable:
 - Form
 - Event

4. If you selected Form Scope and you want to use a grid variable, click the Grid option.
5. Click the data dictionary visual assist to browse data dictionary items.
6. Click the Show All button to display the variable list.
7. Choose the data item to which the variable is associated and click Add.
The system automatically assigns a prefix to the variable, based on the type of scope that you choose.
8. Click OK to return to the design mode for interactive, batch, or business function event rules.
You can now use the new variable.

Creating Event Rules for Automatic Line Numbering

You can create an event rule to automatically number lines.

► To create event rules for automatic line numbering

1. Create a variable to hold the value of the line number. Use the data dictionary item LNID.
`VA frm_LineNumberCounter_LNID`
2. Initialize the variable on the Post Dialog is Initialized event.
`VA frm_LineNumberCounter_LNID = 0`
3. On the Grid Record is Fetched event, number the lines as each line is pulled from the database.
`If BCLineNumber > VA frm_LineNumberCounter_LNID`
`VA frm_LineNumberCounter_LNID = BC LineNumber`
`End If`
4. On the Add Last Entry Row to Grid event, increment the LineNumber and assign the new value to the next available line.
`VA frm_LineNumberCounter_LNID = VA frm_LineNumberCounter_LNID + 1`
`GC LineNumber = VA frm_LineNumberCounter_LNID`

Attaching a System Function to an Event

You can use the System Function button to attach predefined EnterpriseOne system functions to events. For example, you can attach system functions that can do the following to an event:

- Hide or display a control
- Insert dates
- Display media objects

► To attach a system function

1. On Event Rules Design, choose an event.

2. Click the $f(S)$ button.
3. Choose a category in the System Functions box.
4. Choose the system function that you want to attach.
5. In the Available Objects list, choose objects to pass to the system function.
6. Click OK to add the system function to the event rule.

Common System Functions

Following are some available system functions and descriptions of their use.

Control System Functions

The following table describes various control system functions.

Clear Edit Control Error	Use this system function to clear errors that have been set for a specific control. This function is important because the runtime engine automatically sets and clears errors on specific events, but it doesn't clear errors on other events. For example, if you use Set Edit Control Error on the Dialog Is Initialized event, that event cannot run again to reset the error and then re-edit the control. In this situation, you can use Clear Edit Control Error to clear the error on another event.
Disable Control	Use this system function to dynamically enable or disable a control; otherwise, use the control property. You can use this system function to disable certain fields that you do not want a user to change, such as the exchange rate on a transaction.
Enable Control	Use this system function to enable a control. For example, you can use this function when you want only a manager to have the authority to change something.
Hide Control	Use this system function to dynamically hide and show a control. You can hide a field until the user requests the information it holds. For example, when you add a record in Address Book, you can display the previously-added record.
Set Edit Control Error	Use this system function to set an error on a specific control and turn the field red.
Set Status Bar Text	Use this system function to display text in the status bar. This text might describe why an error has occurred and clear the status bar after an error is corrected. You can also use this system function when a process is running in the background or takes some time to complete. In this case, you might want to display a "Please Wait ..." message.
Was Value Entered	Use this system function to verify that data has been changed, such as when an error occurs and the information changes. One of the options for this system function is an All Controls value that determines whether any of the controls have changed. This function is useful in transaction-type programs.

Note

PeopleSoft strongly recommends against using the following control system functions:

- Set Edit Control Color

Set Edit Control Color makes mandatory fields a specific color. If you use this system function, be aware that some users might be colorblind.

- Set Edit Control Font

Set Edit Control Font draws attention to a field, such as a negative number, but nonstandard fonts might confuse the user or detract from the appearance of your form.

See Also

- *Messaging in the Development Tools Guide* for more information about how the runtime engine sets and clears errors

Message System Functions

The message system functions for an event are described as follows:

Mail Send Message Use this function to return a message ID (serial number) that is the identifier for the message that you created. This is the key parameter for the other messaging functions, and the application uses this ID to perform the other system functions. You might use this function to generate a message to a manager that requests approval for a client to exceed an authorized credit limit.

Forward Message Use this function to forward a message. You can choose to leave the original message intact when you forward it.

Mail Delete Message Use this function to delete a mail message.

Update Message Use this function to modify a message with new text. You can then use the Forward Message function to send the new message.

General System Functions

Copy Currency Information Use this function to ensure that, when you change currency information, the system copies the change to all of the applications that use it. This function is useful if you are developing software in a country in which the decimal place changes frequently. You can also use it when a company changes from the base of one currency to another.

Press Button Use this function to reuse event rules from event to event without having to write and maintain the code in all of the places in which you use these event rules. To implement, create a push button on a form and put all of the reusable event rules on the button clicked event. Then use this system function to “press” the button and run the event rule from all of the other events in which you want to run the event rule.

Set Control Focus Use this function to set focus on a specific spot. For example, you might set focus on the detail part of the grid when a user enters a form. You can also use this function to override the default cursor position, particularly if the focus differs between an *add* and a *change*.

Stop Processing	Use this function to stop processing, such as when an error occurs.
Suppress Add	Use this function to control the normal flow of the form for I/O purposes. For example, in transaction entry programs that use master business functions, the master business function performs all of the I/O, including adds.
Suppress Delete	Use this function to control the normal flow of the form for I/O purposes. For example, transactions such as accounts receivable and sales order use master business functions to ensure referential integrity. You might use this function to prevent a user from deleting a customer's address book record that appears on existing invoices.
Suppress Find	Use this function to suppress a find. For example, on updateable grid forms that can use filter and find buttons, you would probably suppress the find button if the user has not finished updating the current record. For those forms that perform inquiries, you can verify that the user has provided information that matches an index.
Suppress Update	Use this function to suppress an update. This function can control the normal flow of the form for I/O purposes.
Was Form Record Fetched	Use this function to ensure that a record was actually fetched.

Grid System Functions

The following table describes the grid system functions for an event:

Clear Grid Buffer	Use this function to clear the buffer. Because the grid buffer is a temporary location in which to manipulate grid rows, use this system function to clear all columns in the buffer. For example, you can use the grid buffer to hold temporary values for a calculation, and then use Clear Grid Buffer to clear the buffer after the system completes the calculation.
Clear Grid Cell Error	Use this function in the same way that you use Clear Grid Buffer.
Clear QBE Column	This function is similar to Clear Grid Buffer. For example, you might use this function when the QBE column had a value forced into it, such as the current date on a purchase order Find/Browse form. Then, after the user performs a Find, you could clear the QBE column in case you want to use another date.
Copy Grid Row to Grid Buffer	Use this function after Clear Grid Buffer was issued and you needed to manipulate just a few of the grid columns. You can use this system function to move all of the GC fields to GB without copying each individual column.
Delete Grid Row	Use this function if you (and <i>not</i> the runtime engine) are manipulating the grid row. For example, use this function when you are using the Delete Doc routine from the master business function.
Disable Grid	Use this function to disable the entire grid from input. Use this function when you need to complete more information in the header before you can add individual rows, or when you want to use information in the header as default information in the grid.
Enable Grid	Use this function to enable the grid.
Get Grid Row	Use this function to specifically read a grid row, such as when you need to reprocess the

grid through a While loop.

Get Max Grid Rows	Use this function to determine how many rows the system must process if you need to reprocess all of the individual grid rows through a While loop. Use this function immediately before the While loop.
Get Selected Grid Row Count	Use this function to get the row number for a selected row. Typically, you use this function only when you need to save the row as a variable for future processing.
Hide Grid Column	Use this function for dynamic processing.
Hide Grid Row	Use this function to hide an entire row from appearing on the form. For example, if you create a form with a grid and you want to summarize rows, add together several rows to determine a total and show the total row. If you use this system function, you might include a check box for the detail information and then unhide the row instead of repopulating the grid.
Insert Grid Buffer Row	Use this function to manipulate the buffer space. If you use this system function, GC becomes GB.
Insert Grid Row	Use this function to insert a row in the grid. Usually, such rows are custom (nondatabase) rows, such as a totaling row.
Set Grid Cell Error	Use this function in the same way that you use Set Edit Control Error.
Set Grid Color	Use this function in the same way that you use Set Edit Control Color.
Set Grid Font	Use this function in the same way that you use Set Edit Control Fonts.
Set Grid Row Bitmap	Use this function to manipulate the bitmap that appears outside of the grid. This function sets a specific bitmap on a specified row header.
Set QBE Column Compare Style	Use this function to assign comparisons to the QBE line. For example, if you want to set up a date column's QBE field to be > January 1, 2005, you would use QC WorkDate = "010105" and Set QBE Column Compare Style(FC Grid, GC WorkDate, <Greater Than>).
Show Grid Column	Use this function in the same way that you use Show Edit Control.
Suppress Grid Line	Use this function to prevent a row from becoming part of the grid. For example, use this system function on the <i>Write Grid Line Before</i> event to prevent the line from being written to the grid.
Update Grid Buffer Row	After the grid buffer is assigned, use this system function to make GC=GB.

Telephony System Functions

You can use telephony system functions to integrate telephone communications with your applications. The telephony system functions allow you to send requests to the Microsoft Windows Telephony Application Programming Interface (TAPI). For example, you might use this capability in a customer service organization that customers call for support. When a customer calls, you can use

your application to identify who is calling and automatically retrieve information about the caller. You can use TAPI to do the following:

- **Answer an incoming call either from a direct line, PBX, or Voice Response Unit (VRU)**
To answer an incoming call, you add event rules at the point at which the phone rings but before the user picks up the receiver, and again immediately after the user picks up the receiver.
- **Transfer a call**
To transfer a call, determine whether the user has requested a transfer, and then add event rules immediately before the request to transfer the call and immediately after the application receives notice that the call has been transferred.
- **Put a call on hold**
To put a call on hold, you add event rules immediately after the application has received notice that the call has been put on hold and immediately after the user has picked up the call that was on hold. The user must be able to see information about all calls that are on hold.
- **Place an outgoing call**
To place an outgoing call, you must pass the phone number being called to the PeopleSoft EnterpriseOne telephony system function, which then uses TAPI to complete the steps necessary to make the call. You add event rules immediately before the call is placed, after the call is answered, and immediately after the call ends.
- **End a call**
Calls can end in several ways, including when the service representative or the caller hangs up, the representative transfers the call, or the system disconnects the call because of a bad phone connection. To properly close the call, you add event rules at the point at which the application receives a message that the call has ended.

Subform System Functions

The following table describes the Subform system functions for an event:

Enable Subform	Use this system function to enable all controls on the Subform to the end user. Parameter: Subform • I/O: Input • Required: Yes Return Values: None Special Handling Instructions and Prerequisites: None Available Events: All Power Browse forms, Power Edit forms, and Subforms events Related Tables: None Related System Function(s): Disable Subform Related Topics: Disable Subform
Disable Subform	Use this system function to disable all controls (except action buttons) on the

Subform. Disabling the Subform makes it read-only.

Parameter: Subform

- **I/O:** Input
- **Required:** Yes

Return Values: None

Special Handling Instructions and Prerequisites: None

Available Events: All Power Browse forms, Power Edit forms, and Subforms events

Related Tables: None

Related System Function(s): Enable Subform

Related Topics: Enable Subform

Hide Subform

Use this system function to hide Subforms.

Parameter: Subform

- **I/O:** Input
- **Required:** Yes

Return Values: None

Special Handling Instructions and Prerequisites: None

Available Events: All Power Browse forms, Power Edit forms, and Subforms events

Related Tables: None

Related System Function(s): Show Subform

Related Topics: Show Subform

Show Subform

Use this system function to show Subforms.

Parameter: Subform

- **I/O:** Input
- **Required:** Yes

Return Values: None

Special Handling Instructions and Prerequisites: None

Available Events: All Power Browse forms, Power Edit forms, and Subforms events

Related Tables: None

Related System Function(s): Hide Subform

Related Topics: Hide Subform

Update Parent

Use this system function to update the mapping link values between the calling Subform and its parent. Items that are mapped both bi-directionally (child-to-parent and parent-to-child) and in a single direction (from child to parent) are updated because of this system function call.

Parameter: None

- **I/O:** None
- **Required:** No

Return Values: None

Special Handling Instructions and Prerequisites: This system function updates the mapping link values between the form that calls the system function and its parent form. The parent form can be a Power Browse form, Power Edit form, or another Subform. Both items that are mapped bi-directionally (child-to-parent and parent-to-child) and in a single direction (from child to parent) are updated because of this system function call. The latest Subform Interconnect (SI) values on the Subform are copied to the designated form controls on the parent form.

Available Events: All Power Browse forms, Power Edit forms, and Subforms events

Related Tables: None

Related System Function(s): None

Related Topics: Notified Child and Notify Parent

Notify Parent

Use this system function to trigger the Notified By Child event to run on the parent form of the calling Subform.

Parameter: None

- **I/O:** None
- **Required:** No

Return Values: None

Special Handling Instructions and Prerequisites: Mapping link values are not updated automatically. To update mapping links, use system function Update Parent. You should only use this system function on a Subform.

Available Events: All Subform events

Related Tables: None

Related System Function(s): Update Parent updates the mapping link values from the calling Subform to its parent.

Related Topics: Update Parent

Get Error Count

Use this system function to return the number of errors on the Subform.

Parameter: Subform

- **Description:** The Subform on which the errors occur.
- **I/O:** Input
- **Required:** Yes

Parameter: Number

- **Description:** The number of errors on the Subform on which the warnings occur.
- **I/O:** Output
- **Required:** Yes
- **Note:** This output parameter must have either an integer or a math numeric data dictionary type.

Return Values: Number; the number of errors on the Subform.

Special Handling Instructions and Prerequisites: None

Available Events: All Power Browse forms, Power Edit forms, and Subforms events

Related Tables: None

Related System Function(s): Get Warning Count

Related Topics: Get Warning Count

Get Warning Count

Use this system function to return the number of warnings on the Subform.

Parameter: Subform

- **Description:** The Subform on which the warnings occur.
- **I/O:** Input
- **Required:** Yes

Parameter: Number

- **Description:** The number of warnings on the Subform on which the warnings occur.
- **I/O:** Output
- **Required:** Yes
- **Note:** This output parameter must have either an integer or a math numeric data dictionary type.

Return Values: Number; the number of warnings on the Subform.

Available Events: All Power Browse forms, Power Edit forms, and Subforms events

Related Tables: None

Related System Function(s): Get Error Count

Related Topics: Get Error Count

Get Subform ID

Use this system function to return the unique control ID of the calling Subform. You should only use this system function on a Subform.

Parameter: Subform ID—holds the ID of the calling Subform

- **I/O:** Output
- **Required:** Yes
- **Notes:** The output parameter must have a numeric data type, including integer, handle, and math numeric.

Return Values: Returns the current Subform control ID.

Special Handling Instructions and Prerequisites: You should only use this system function on a Subform.

Available Events: All Subform events

Related Tables: None

Related System Function(s): None

Related Topics: None

Notify Child

Use this system function to notify a child Subform and trigger the Notified By Parent event to run on the child Subform.

Parameter: Subform

- **I/O:** Input
- **Required:** Yes

Return Values: None

Special Handling Instructions and Prerequisites: You should only use this system function on a Power Form or a Subform that is a parent to another Subform.

This system function first updates the mapping link. All items that are mapped from parent to child are updated with the latest parent control values.

Second, the Notified By Parent Event Rule executes for the child Subform.

Available Events: All Power Browse forms, Power Edit forms, and Subforms events

Related Tables: None

Related System Function(s): None

Related Topics: None

Trigger Default Child Action

Use this system function to call an action button on the child form. As a result, the default runtime processing for that button and the event rules for the button are executed.

Parameter: Subform

- **Description:** The child Subform whose action button the system calls.
- **I/O:** Input
- **Required:** Yes

Parameter: Default Action

- **Description:** The default action button that the system calls.
- **I/O:** Input
- **Required:** Yes
- **Note:** For Subforms on Power Edit forms, valid values are:
 - Save
 - Delete
 - Find
 - Clear
 - Select

For Subforms on Power Browse forms, valid values are:

- Find
- Select

Return Values: None

Special Handling Instructions and Prerequisites: If the action button does not exist on the child form, no action is taken.

This system function is called from the parent form, which can be a Power

Browse form, a Power Edit form, or a Subform.

The end result depends on the designated action button. Before any of the default action button Event Rules (ER) are executed, the Subform Interconnect (SI) values update with the latest parent control values.

Available Events: All Power Browse forms, Power Edit forms, and Subforms events

Related Tables: None

Related System Function(s): None

Related Topics: None

Parent/Child System Functions

The parent/child system functions for an event are described below:

Insert Grid Buffer Row By Node ID

Use this system function to insert a tree node at any position.

Parameters:

1. Parent/Child Control

- **I/O:** Input
- **Required:** Yes

2. Base On Node ID: the ID of the referenced node on which a new node will be inserted

- **I/O:** Input
- **Required:** Yes

3. Insert As: Insert location. The relationship of the newly inserted node to the referenced node.

- **I/O:** Input
- **Required:** yes
- **Valid Values:** <As First Child>, <As Last Child>, <As Previous Sibling>, <As Last Sibling>

4. Selectable: whether user can select the newly inserted node and row.

- **I/O:** Input
- **Required:** yes

5. Protected: whether user can modify the newly inserted node and row.

- **I/O:** Input
- **Required:** yes

6. Updatable: whether the system should try to update the database with the new row.

- **I/O:** Input
- **Required:** yes

7. Delete-able: whether user can delete the newly inserted node and row.

- **I/O:** Input
- **Required:** yes

8. Clear After: whether the system will clear the grid buffer after this system function call.

- **I/O:** Input
- **Required:** yes
- **Valid Values:** Boolean

9. Expandable: whether the user can expand the newly inserted node and row. If the row is not expandable, then the node will be a leaf node.

- **I/O:** Input
- **Required:** yes

10. Insert Mode: if the parent/child control has a business view, this parameter tells the system if the new row is to be inserted into the database or to be updated into the database (assuming such a row already exist in the database). If the insert mode is UPDATE and there is no such row in the database, the system will fail to process this row. If the insert mode is INSERT and there is already a row in the database, the system will fail to process the row in the database.

- **I/O:** Input
- **Required:** yes
- **Valid Values:** <Insert>, <Update>

Return Values: None

Special Handling Instructions and Prerequisites:

Available Events: All events

Related Tables: None

Related System Function(s): Get Related Node ID

Related Topics: Get Related Node ID

Get Related Node ID

Use this system function to find out the ID of a node related to a reference node.

Parameters:

1. Parent/Child

- **I/O:** Input
- **Required:** Yes

2. Current Node ID: the ID of the reference node

- **I/O:** Input
- **Required:** Yes

3. Relation: the relation of the target node to the reference node.

I/O: Input

- **Required:** Yes
- **Valid Values:** <Parent Node>, <First Child>, <Next Sibling>, <Previous Sibling>

4. Node ID: the ID of the target node

- **I/O:** Output
- **Required:** Yes

5. Error Code: 1 if the system can not find the reference node or the target node. 0 if the system function is successful.

- **I/O:** Output
- **Required:** Yes

Special Handling Instructions and Prerequisites: None

Available Events: All events

Related Tables: None

Related System Function(s): Insert Grid Buffer Row By Node ID

Related Topics: Insert Grid Buffer Row By Node ID

Disable Grid

Use this system function to disable a cell, a tree node, a row, a column or the entire parent/child control.

Parameters:

1. Parent/Child

I/O: Input

Required: Yes

2. Row: which row is to be disabled.

I/O: Input

Required: Yes

3. Column: which column is to be disabled.

I/O: Input

Required: Yes

Special Handling Instructions and Prerequisites: None

Available Events: All events

Related Tables: None

Related System Function(s): Disable Grid under Grid System Functions

Related Topics: Disable Grid under Grid System Functions

Set Grid Cell Error

Use this system function to set an error on a cell, a tree node, a row, a column or the entire parent/child control.

Parameters:

1. Parent/Child

I/O: Input

Required: Yes

2. Row: which row to set error on.

I/O: Input

Required: Yes

3. Column: which column to set error on.

I/O: Input

Required: Yes

4. Error Code: the alias of the error to be set on the row and column. **I/O:** Input

Required: Yes

Special Handling Instructions and Prerequisites: None

Available Events: All events

Related Tables: None

Related System Function(s): Set Grid Cell Error under Grid System Functions

Related Topics: Set Grid Cell Error under Grid System Functions

Clear Grid Cell Error

Use this system function to clear an error on a cell, a tree node, a row, a column or the entire parent/child control.

Parameters:

1. Parent/Child

I/O: Input

Required: Yes

2. Row: which row to clear error on.

I/O: Input

Required: Yes

3. Column: which column to clear error on.

I/O: Input

Required: Yes

Special Handling Instructions and Prerequisites: None

Available Events: All events

Related Tables: None

Related System Function(s): Clear Grid Cell Error under Grid System Functions

Related Topics: Clear Grid Cell Error under Grid System Functions

Was Grid Cell Value Entered

This system function returns if a specific grid cell or tree node has been changed since last time this system function is called.

Parameters:

1. Parent/Child

I/O: Input

Required: Yes

2. Column: the column of the grid cell we are checking.

I/O: Input

Required: Yes

Special Handling Instructions and Prerequisites: None

Available Events: All events

Related Tables: None

Related System Function(s): Was Grid Cell Value Entered under Grid System Functions

Related Topics: Was Grid Cell Value Entered under Grid System Functions

Attaching a Business Function to an Event

You can attach an existing business function to an event. Business functions include:

- C code that you generate manually (source language C)
- Code that PeopleSoft EnterpriseOne software generates when you use Business Function Event Rule Design (source language NER)

You typically use business functions for the following purposes:

- Referential integrity, such as deleting secondary records when a master record is deleted, and for editing routines
- Large and complex calculations that might otherwise overload the runtime engine

► To attach a business function

1. On Event Rules Design, choose an event.
2. Click the *f*(B) button.

You can view a description for a business function by choosing Attachments from the Row menu.

3. Choose a business function and click Select.
4. In the Available Objects list, choose objects to pass to the business function.
5. To assign a literal to the business function, choose <Literal> in the Available Objects list.
6. Enter a single value and click OK.

Range and List are not valid literals to use with business functions.

7. Indicate the direction of data flow between Value and Data Items, and click OK.

As you click the direction arrow, it toggles through the following four options:

- ! Data flows from the source to the target
- z Data flows from the target to the source
- \$ Data flows from the source to the target, and upon exiting the target, data flows back to the source
- No data flow

If the direction of the items is hard-coded in the data structure for a business function (such as when the parameters are predetermined to be input, output, or bidirectional), then this predetermined direction appears here. If the direction of items is not critical, it can be omitted or set by the user. You must complete the required items that appear in red. The status bar indicates the state of the flow to the target.

8. Click the Include in Transaction option to include this business function for transaction processing.

This option appears only if you are calling from a Fix/Inspect, Header Detail, or Headerless Detail form

9. Click the Asynchronously option to enable asynchronous processing.

This option appears for the Post Button Clicked event, or for any Cancel or OK button on a Fix/Inspect form, Header Detail form, or Headerless Detail form.

10. Click one of the following buttons to add notes.

- Business Function Notes
- Structure Notes
- Parameter Notes

11. Click OK.

EnterpriseOne Business Function Design Standards

Always use the directional arrows to attach business functions. If you do not use a parameter, then use the  symbol. This symbol also does the following:

- Identifies when a parameter has been forgotten or needs to be added, such as when a business view changes or a work field is deleted
- Serves as documentation to other readers of the code
- Prevents memory from being reserved when it is not needed

Because they are internationally acceptable, use numeric values for any event rule flags that are passed back from the business functions. For example, to assign true/false values, use 1 for true and 0 for false, instead of T and F or Y and N.

See Also

- ❑ *Business Functions* in the *Development Tools Guide* for more information about business functions

Creating Form Interconnections

Use form interconnections to automatically pass data from a source form to a target form.

Prerequisite

- ❑ Define the data structure of the receiving form to include any field for which you are passing values.

Creating a Modal Form Interconnection

Most form interconnections are modal. This situation means that after a form interconnects to another form, the user must close the second form before interacting with the first form again.

► To create a modal form interconnection

1. On Event Rules Design, choose an event.
2. Click the Interconnect button.
3. On Work with Applications, choose the application to which you are connecting.

Work with Forms displays available forms for the chosen application.

4. Choose the appropriate form to which you want to connect (the target).
5. Choose the appropriate version of the form to which you want to connect.

The Data Item column displays data items in the data structure of the target form. The keys in the primary unique index for the primary table of the business view are automatically set up as the data structure.

6. In the Available Objects column, choose the object that you want to pass.
7. Use the > button to move it to the Data Structure-Value Column.

Indicate the direction of data flow between Value and Data Items.

If you do not want data to pass between forms, set all Direction values to null, and then click OK to save the Form Interconnection and exit.

As you click the direction arrow, it toggles through the following five options:

- Data flows from the source to the target
 - Data flows from the target to the source
 - Data flows from the source to the target and, upon exiting the target, data flows back to the source.
 - Upon exiting the target, data flows back to the source
 - No data flows either way
8. Click the Include in Transaction option to include this interconnection for transaction processing.

This option appears only if you are calling from a fix/inspect, header detail, or headerless detail form.

9. Click one of the following buttons to add notes:

- Structure Notes
- Parameter Notes

10. After the data structure is defined, click OK.

Event Rules Design displays the Form Interconnection with the following statement:

Call (Application <name> Form <name>)

EnterpriseOne Design Standards for Form Types

The following standards apply to all form types:

- Accept the default placement of primary unique key fields at the top of the data structure.
- Change the data item name and description to describe the item that is passed between forms.

Creating a Modeless Form Interconnection

You can create a modeless form interconnection between a find/browse form and one or more fix/inspect forms, or between a find/browse form and transaction forms (header and headerless detail forms).

Modeless processing allows the user to interact with two or more forms simultaneously instead of having to close one form to return to the other. When you update the transaction form, you do not need to re inquire on the database before the update appears on the find/browse form. After accessing the initial find/browse, you can access any form type, including another find/browse.

Each time that the system calls a modeless form interconnection, it passes the values in the form data structure. When you click Cancel in the calling form, the system destroys the called form and does not pass any values. When you click OK in the called form, the system passes values back to the calling form through the form data structure. The called form still appears.

The *Dialog is Initialized* event occurs only the first time that the system calls it. If you want event rules to run each time a form appears, attach them to *Post Dialog is Initialized*.

If a form in update mode fails to fetch a record from the database, the form mode changes to Add mode. If the *Close Form On Add* option is turned on, the form closes when you click OK.

When you close a Find/Browse form, it closes all of the modeless interconnect records before it closes.

Event rules that follow a modeless form interconnection execute immediately instead of waiting for the called form to return.

You run applications with modeless processing from the menu. If you run an application from Object Librarian, modeless processing does not work.

► To create a modeless form interconnection

1. On Event Rules Design for the find/browse form that you want to use (the source), choose an event.
2. Click the Form Interconnection button.

3. On Work with Applications, choose the application to which you are connecting.
Work with Forms displays available forms for the chosen application.
4. Choose the appropriate fix/inspect form to which you want to connect (the target).
The Form Interconnect - Values to Pass window displays the data structure for the target form.
5. Choose the appropriate versions of the fix/inspect form to which you want to connect.
The Data Item column displays data items in the data structure of the target form. The keys in the primary unique index, for the primary table of the business view, are automatically set up as the data structure.
6. Click the Modeless option.
7. In the Available Objects column, choose objects that you want to pass. Use the > button to move objects to the Data Structure-Value Column.
8. Indicate the direction of data flow between Value and Data Items.
If you do not want data to pass between forms, set all values to ← and click OK to save the Form Interconnection and exit.

As you click the direction arrow, it toggles through the following four options:
 - Data flows from the source to the target
 - Data flows from the target to the source
 - Data flows from the source to the target and, upon exiting the target, data flows back to the source
 - No data flow
9. Click one of the following buttons to add notes:
 - Structure Notes
 - Parameter Notes
10. After you define the data structure, click OK.
The Event Rules Design displays the Form Interconnection with the following statement:

Call (Application <name> Form <name>).

Prerequisite

- ☐ Set up the data structure of the receiving form to include any field for which you are passing values.

Creating Report Interconnections

Use report interconnections to automatically run a report. The current event rules might continue processing or wait for the completion of the report, based upon whether asynchronous processing is enabled. If you use a synchronous report interconnect, your initiating process starts a second process

and waits until the second process has completed before it continues running. If you use asynchronous processing, the initiating process starts another process and continues to run. The two processes run separately.

► To create a report interconnection

1. On Event Rules Design, choose an event.
2. Click the Report Interconnection button.
3. On Work with Applications, choose the report to which you are connecting.
Work with Versions displays versions for the selected report.
4. Choose the appropriate version of the report to which you want to connect.
5. In the Available Objects column, choose the object that you want to pass. Use the > button to move it to the Data Structure-Value Column.
6. Indicate the direction of data flow between Value and Data Items.
If you do not want data to pass between reports, set all Direction values to ← and click OK to save the Report Interconnection and exit.
As you click the direction arrow, it toggles through the following four options:
 - Data flows from the source to the target
 - Data flows from the target to the source
 - Upon exiting the target, data flows back to the source
 - No data flow
7. To run the report as a separate process, click the following option:
 - AsynchronouslyYou must turn on this option if you want to pass data into the report.
8. To include the report interconnect for transaction processing, click the following option:
 - Include in Transaction
9. Click one of the following buttons to add notes:
 - Structure Notes
 - Parameter Notes
10. After the data structure is defined, click OK.
Event Rules Design displays the Report Interconnection with the following statement:
 - Call (UBE <name> Version <name>)

Creating Assignments

Use an assignment to define a field as a fixed value or a mathematical expression. For example, you can create an assignment that inserts a default value when the user leaves the field. You can also use an assignment to calculate a value, rather than creating a business function to calculate the value.

When you create an assignment, you can do the following:

- Assign a fixed value.
- Create and assign a mathematical expression.

When you create an expression, calculate only the data items that are of the exact same numerical scale or data type. For example, do not calculate different currencies or decimal figures that represent different decimal values because the result of these calculations might compromise data integrity.

► To assign a value

1. On Event Rules Design, choose an event.
2. Click the Assignment/Expression button.
3. On Assignment, choose the To Object that you want to receive your assigned value.
4. Use one of the following methods to determine the From/Object Literal value:
 - Choose a From Object in the right-hand column to create a simple statement: [left-hand column] = [right-hand column].
 - Type a literal expression (number, text, and so on) in the text entry box to assign a literal statement: [left-hand column] = [literal].
 - Press the $f(X)$ button to create a complex expression or advanced mathematical function using the Expression Manager.
5. Click OK.

The Event Rules Design form displays the assignment in text format.

► To create an expression for an assignment

1. From the Assignment form, click the $f(X)$ button.
2. On Expression Manager, create the expression in the edit field.

Either type the expression or use the object list, calculator pad, and functions list. You can use the following buttons to edit your expression:

Clear Clears all data from the Assignment Display

Bksp Deletes the previous expression or character

Undo Reverses your previous editing action (other than inserting objects or functions from the lists)

3. Click OK when you finish editing your expression.

The Assignment Display checks for and highlights any spelling or syntax errors. If it detects no errors, the system returns the expression to Event Rules Design.

Expression Manager

Expression Manager has several different sections with which you can work, as described in the following topics.

Assignment Display

The first section of the Expression Manager form displays the current status of the assignment expression. To Object displays the object to which the expression will be assigned. From Expression displays either a blank entry or the expression currently being edited.

Expression Editing Field

You create and edit expressions in the editing field. You can enter text by typing characters from the keyboard, using the editing keypad, or using the mouse to point and click to various expression components, including the following:

- Objects
- Numbers and mathematical symbols
- Advanced functions

The editing field accepts any expression, in any order. If syntax errors occur, the system highlights the error and displays it in the status bar.

Available Information

The Available Information list contains all of the objects that are available for creating and editing expressions. When you choose an object in the list box, the system displays its data type in the status bar.

Editing Keypad

You can use the editing keypad to enter data into an expression instead of using the keyboard. The Undo button is enabled only when the system can undo the last editing operation that you performed. The system cannot undo inserts from either the Available Information list or the Advanced Functions list.

Advanced Functions

The Advanced Functions list contains functions that are available to create or edit an expression. You can display information about functions in the status bar. The first box displays a short description of the function, and the second box displays the syntax for the function. Advanced functions include:

- Date functions
- General functions
- Trigonometry functions
- Text functions

Date Functions

Function	Description
add_days(date, days)	Adds a specified number of days to the date
add_months(date, months)	Adds a specified number of months to the date
days_between(date1, date2)	Returns the number of days between date1 and date2
months_between(date1, date2)	Returns the number of months between date1 and date2

date_day(date)	Returns the day for the specified date
date_month(date)	Returns the month of the year for the specified date (1=January, 2=February,...)
date_today()	Returns the current date
date_year(date)	Returns the year for the specified date (in 19xx format)
last_day(date)	Returns the last day of the month of the specified date (not day of the week)
next_day(date, day_of_week)	Returns the next date that occurs on the day of the week after a given date

General Functions

Function	Description
abs	Returns the absolute value of the specified number
cell	Rounds the specified number up to the next whole number
exp	Calculates the exponential e to the specified number
floor	Rounds the specified number down to the next whole number
mod	Returns the remainder of numeric1/numeric2
pow10	Calculates 10 to the power of the specified number
pow	Calculates numeric1 to the power of numeric2
round	Rounds numeric1 to the power of numeric2
sign	Returns the sign of the specified number (If the sign is negative, the system returns a -1; otherwise, it returns 0.)
sqrt	Calculates the positive square root of the specified number

Trigonometry Functions

Function	Description
acos	Calculates the arc cosine of the specified number
asin	Calculates the arc sine of the specified number
atan2	Calculates the arc tangent of numeric1/numeric2
cos	Calculates the cosine of the specified number

cosh	Calculates the hyperbolic cosine of the specified number
log	Calculates the natural logarithm of the specified number
log10	Calculates the base 10 logarithm of the specified number
sin	Calculates the sine of the specified number
sinh	Calculates the hyperbolic sine of the specified number
tan	Calculates the tangent of the specified number
tanh	Calculates the hyperbolic tangent of the specified number

Text Functions

Function	Description	Example
concat	Appends string2 to string1	
indent	Indents string a given length of characters	indent("abcde",5,5)="+++++abcde"
length	Returns the length of a string	length("axcvbnm") = 7
lower	Converts specified string to lowercase	lower("AbCdEfG") = "abcdefg"
lpad	Inserts the character into the front (left side) of the string until the string is length long	lpad("qwerty",' ',12) = ".....qwerty"
ltrim	Removes leading occurrences of the character from the string	ltrim("aaaaaabcdef" , 'a') = "abcdef"
rpadd	Inserts the character into the back (right side) of the string until the string is length long	rpadd("qwerty" , '.', 10) = "qwerty...."
rtrim	Removes trailing occurrences of the character from the string	rtrim("abcdef" , ' ') = "abcdef"
substring	Extracts a portion of string beginning at position for length number of characters Note The beginning position of the string is considered position zero; to substring beginning with the 10th character, use 9 as the beginning position in the command.	substring ("SUNMONTUEWEDTHUFRISAT", 9,3) = "WED"
upper	Converts specified string to uppercase	upper("qweRTY") = "QWERTY"

Table I/O

Use the Table I/O button in Event Rules Design to create instructions that perform table input and output (I/O) so that you do not need to manually code a business function in C code. Table I/O allows you to access a table through event rules. You can use table I/O to do the following:

- Validate data.
- Retrieve records.
- Update or delete records across files.
- Add records.

For example, you can use table I/O to display information in a table that your application does not use.

You can use Log Viewer to view your table I/O SQL statements in the `jddebug.log`. To do so, your `jde.ini` file must have debugging set to File.

Available Operations

Table I/O is an event rule that replaces writing low-level business functions that perform I/O by calling to the JDB API set. Using table I/O, you can perform the following operations:

- FetchSingle** Combines Select and Fetch in a basic operation. Indexed columns are used for the Select; and non-indexed columns are used for the Fetch. The operation opens a table for I/O but does not close it. All tables that do not use handles close automatically when the form that uses them closes.
- Insert** Inserts a new row.
- Update** Updates an existing row. Only those columns mapped (presently in tables with or without handles) are updated. You can do partial key updates with tables and handles to tables. If you do not specify all the keys, then several records might be updated.
- Delete** Deletes one or more rows in a table or business view.
- Open** Opens a table or business view.
- Close** Closes a table or business view.
- Select** Selects one or more rows for a subsequent FetchNext operation.
- SelectAll** Selects all rows for a subsequent fetch next operation.
- FetchNext** Fetches rows that you specify. You can fetch multiple records with multiple FetchNext operations or with a FetchNext operation in a loop.

Valid Mapping Operators

You can use the following operators for mapping specific table I/O operations:

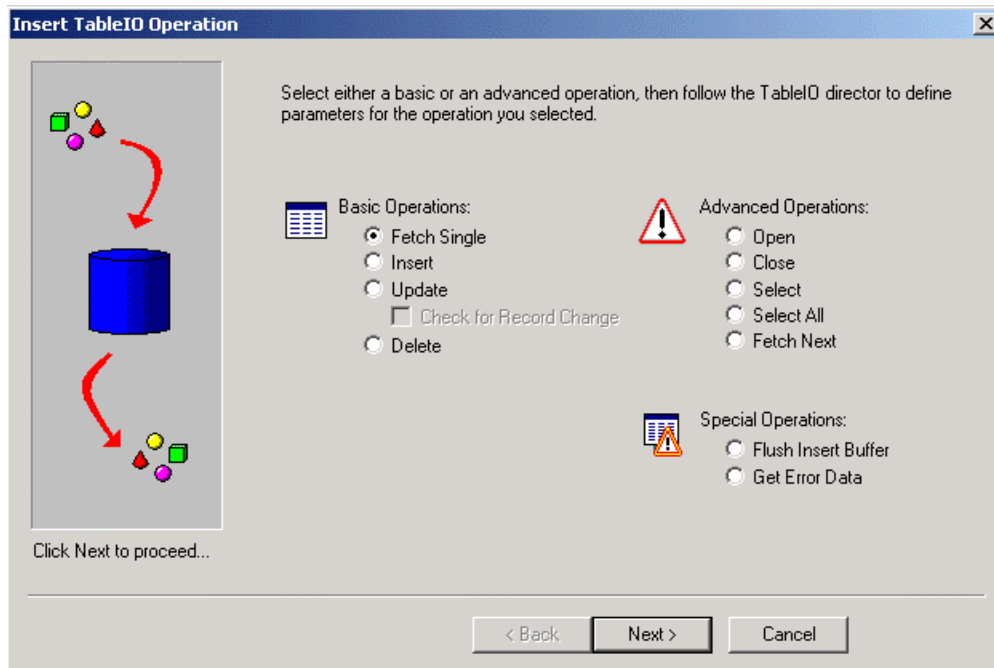
Table I/O Operation	Mapping Operators
FetchSingle	Index Fields: =, <, <=, >, >=, !=, Like Non-Index Fields: Copy Target
Insert	All Fields: Copy Source
Update	Index Fields: = Non-Index Fields: Copy Source
Delete	All Fields: =
Open	N/A
Close	N/A
Select	All Fields: =, <, <=, >, >=, !=, Like
SelectAll	N/A

Creating a Table I/O Event Rule

Table I/O event rules allow event rule support for database access. You need event rule support to perform table input and output, data validations, and record retrieval.

► To create a table I/O event rule

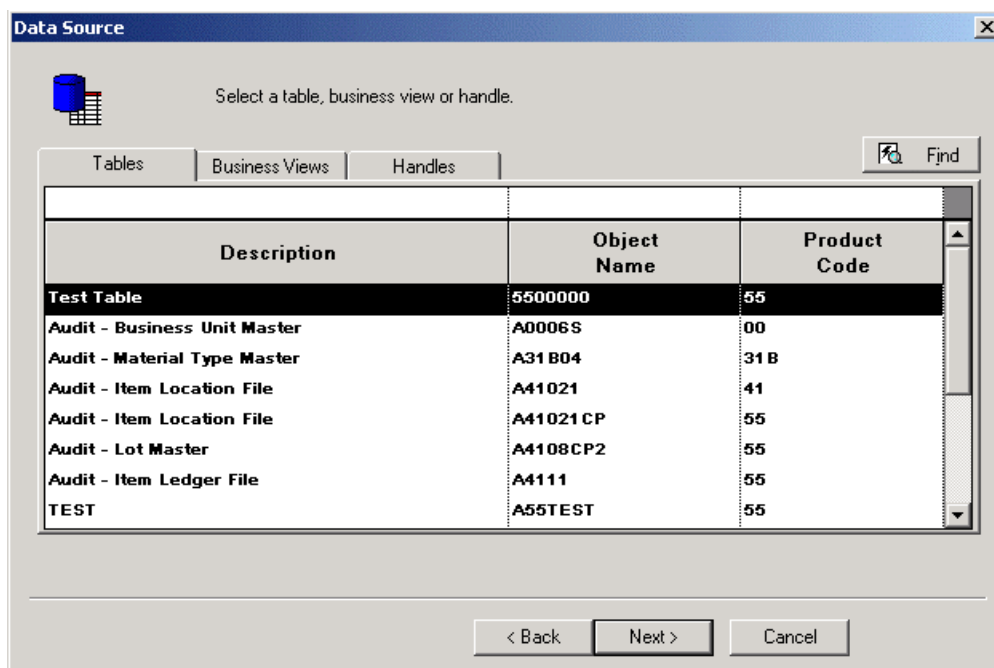
1. On Event Rules Design, click the Table I/O button.



The **Insert TableIO Operation** dialog box is shown. It contains a diagram on the left showing data flow between a database cylinder and a set of colored shapes. The main area has instructions: "Select either a basic or an advanced operation, then follow the TableIO director to define parameters for the operation you selected." There are three sections of radio buttons: Basic Operations (Fetch Single, Insert, Update, Check for Record Change, Delete), Advanced Operations (Open, Close, Select, Select All, Fetch Next), and Special Operations (Flush Insert Buffer, Get Error Data). Navigation buttons at the bottom are < Back, Next >, and Cancel.

Click Next to proceed...

2. On Insert TableIO Operation, choose an operation and click Next.



The **Data Source** dialog box is shown. It has tabs for Tables, Business Views, and Handles. A table lists available data sources with columns for Description, Object Name, and Product Code. A Find button is in the top right. Navigation buttons at the bottom are < Back, Next >, and Cancel.

Description	Object Name	Product Code
Test Table	5500000	55
Audit - Business Unit Master	A0006S	00
Audit - Material Type Master	A31B04	31B
Audit - Item Location File	A41021	41
Audit - Item Location File	A41021CP	55
Audit - Lot Master	A4108CP2	55
Audit - Item Ledger File	A4111	55
TEST	A55TEST	55

3. On Data Source, choose the table, business view, or handle for which you want to perform I/O, and then click Next.

The Mapping form displays available objects that you can map to selected table columns. The available objects are used to build a WHERE clause on SELECT statements. For FETCH statements, the system maps values into available objects in the variable column. For example, if you want to FETCH a single record from a table, map columns to create the WHERE clause of a SELECT statement.

Key columns have an asterisk next to them.

Mapping

Select an index and define mappings for this operation.

Address Book Master
F0101.FetchSingle

Index:
Address #

Mapping:

Column Name	Operator	Mapping
* AddressNumber	[Operator]	
AlternateAddressKey	[Operator]	
TaxId	[Operator]	
NameAlpha	[Operator]	
DescripCompressed	[Operator]	
CostCenter	[Operator]	
StandardIndustryCode	[Operator]	
LanguagePreference	[Operator]	

Available Objects

- Business View Column
 - BC Item Number - Short
 - BC Time of Day (F4601)
 - BC User Reserved Amount
 - BC User Reserved Number
- Previous Business View Column
- System Value

< Back Finish Cancel

4. Choose the column that you want to use, and then double-click the available objects that you want to map to that column.
5. Click the Operator button until you locate the operation that you want.

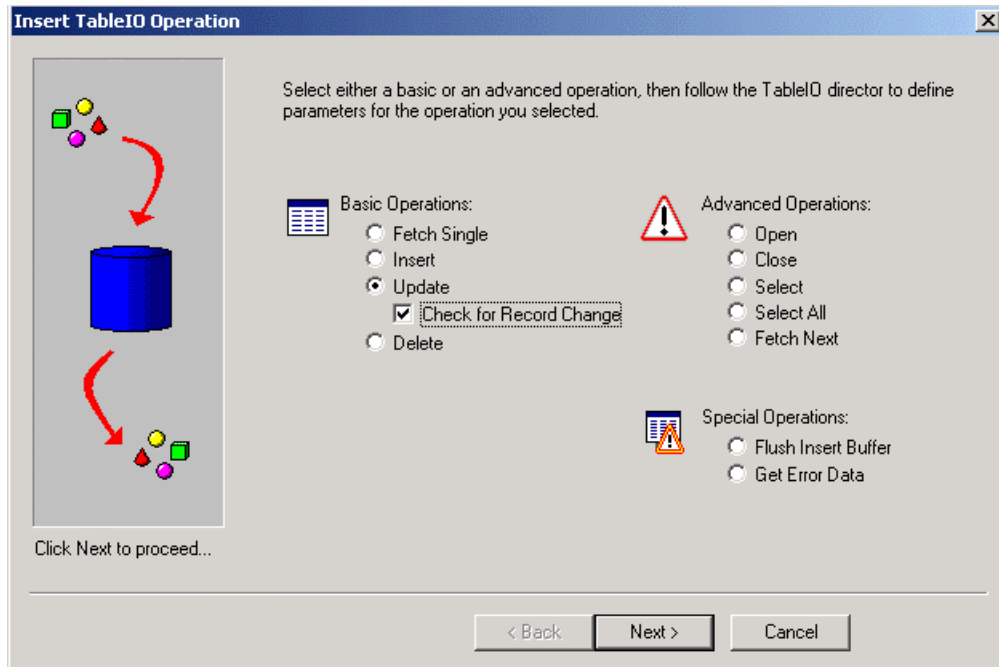
The following selection operations are available. The default is equal.

- Equal
- Not Equal
- Less Than
- Less Than or Equal To
- Greater Than
- Greater Than or Equal
- Like

6. Click Finish to save the operation and return to Event Rules Design.

Record Change

If you choose the Update option, you can also enable Check for Record Change. The Check for Record Change option indicates whether the record has been changed between the time that it was fetched and the time that it was updated.



Buffered Inserts

You can use buffered inserts to improve performance when you insert hundreds or thousands of records into a single database table and you don't need immediate user feedback if an insertion failure occurs. You can use buffered inserts with table conversion, table I/O, batch processes, and business functions, but they are not available for interactive applications. Buffered inserts are available only with Oracle (V8 and above), DB2/400, and SQL Server. Buffered inserts are not available with Access, post-insert triggers, or multiple-table views. The JDEbase database middleware delivers records to the database management system one bufferload at a time.

When you request buffering, the database records are inserted individually and the buffer is automatically flushed when it fills; that is, the JDEbase database middleware delivers the buffer to the database management system. The buffer can also be explicitly flushed. For example, the buffer flushes automatically when you commit a transaction or when you close a table or view. The business function, table conversion engine, or table I/O can explicitly request that the buffer be flushed as well.

Buffered Insert Error Messaging

Because the system provides no immediate feedback if an insertion fails, you should use caution when you decide to use buffered inserts. If an insertion fails, the error appears in the JDE log. Consequently, buffered inserts are used primarily with batch applications.

Unless you are using the Table Conversion application, you must request more detailed information from the middleware to get detailed error messages. In Table Conversion, the table conversion engine automatically performs this task. You can enable tracing to receive more detailed error messages.

Otherwise, you get an error message that the insert failed. You should clear the output tables so that you do not receive duplicate error logging.

► **To use buffered inserts in table I/O**

1. On Event Rules Design, click the table I/O button.
2. On Insert TableIO Operation, choose the Open option in Advanced operations.
3. Click Next.
4. On Data Source, choose the table that you want to use, and then click Advanced Options.
5. On Advanced Options, choose Buffered Inserts, and then click OK.

Using Special Operations for Buffered Inserts

After you set up buffered inserts, you can use Special Operations to flush the buffers or get error messages.

► **To use special operations for buffered inserts**

1. In Event Rules, at the point at which you want to perform either a Flush Insert Buffer or Get Error Data operation, click one of the following options:

- **Flush Insert Buffer**

To maintain data integrity, you should flush the insert buffer before you perform any operations other than an insert. If you fail to do so, the results of recent inserts might not be reflected in other operations, and the operations might not work properly.

When you use the flush insert buffer option for a specific table, you must flush the buffer before you close the table so that you still have access to error information. You can use the Get Error Data option for each insert to ensure that you can access this information.

- **Get Error Data**

The Get Error Data option retrieves errors for records that the system did not insert properly. Depending on when your buffers are flushed, or when you begin another insert, you might overwrite the error information for a specific insert. If error information is critical, retrieve the information before the next insert begins.

If you need to perform special error handling, you should set it up after each table I/O insert and each Flush Insert Buffer option. Always retrieve the error information before you begin the next table operation.

Note Concerning `CO_ERROR_DETAILS_AVAILABLE` for *SV_File_IO_Status* for Reports

You set this error for reports during an Insert or Flush Insert Buffer operation when the insert fails in Table I/O. You can use the return code to determine if the system provides information about why this insert failed. If the return code from the insert is `CO_ERROR_DETAILS_AVAILABLE`, then you can call the Get Error Data Table I/O Operation. The Get Error Data operation returns the values used in the insert for all the requested columns. The following example illustrates how you can use `CO_ERROR_DETAILS_AVAILABLE`.

```

F0101.Insert  //Attempt an insert

    SL AgingDaysAP1 -> TK Address Number

    RC Page - -> TK Tax ID

    SL TargetEnvironment -> TK Description - Compressed
If SV File_IO_Status is equal to CO ERROR_DETAILS_AVAILABLE
    F0101.Get Error Data  //Failed with errors so get errors
                                //Map values used in insert to the
                                //specified fields.

    SL AgingDaysAP1 <- TK Address Number

    RC TESTT <- TK Tax ID

    SL ReportName <- TK Description - Compressed

End If

```

Handles

In PeopleSoft EnterpriseOne software, the table I/O term handle refers to a type of file pointer. This file pointer connects the application or UBE with the middleware that communicates with the database manager. Handles point to a database table, and they are references to an address within the middleware. Unlike regular file pointers, handles allow you some control over when and how they are used. Handles allow you to perform the following operations, which you cannot perform using nonhandle table I/O operations:

- Concurrently open multiple instances of a single table or business view
- Open a table or business view in an environment other than the environment that you signed on to

This feature is particularly helpful when you receive an upgrade to PeopleSoft EnterpriseOne software or when you need to convert data from another system into PeopleSoft EnterpriseOne software.

- Pass handles into a form, named event rule, or business function so that you do not need to open a table or business view more than once

Note

You cannot use handles in transaction processing.

If you pass a handle to a form or a named event rule, the data structure for the form or named event rule must contain a member that is a handle data item. In the form interconnect or business function call, you must assign a handle value from your event rules to the handle data structure member. You can use this handle in the form or named event rule that is receiving the handle in the same way that you use any other handle.

You must explicitly open and close handles, unlike other table I/O operations in which you can use implicit open and close statements. You must open a handle before you can use it for any other operations. All of the operations except Open work the same for handles as they do for tables or business views. When you are finished using a handle, you must explicitly close it. You close the handle in the same way that you close a table or business view except that you choose a handle instead of a table or business view.

Using a Handle

To use a handle, you must do the following:

- Define the handle in the data dictionary.
- Create a handle variable in event rules.
- Open the handle explicitly.

After you create a handle data item, you must create a handle variable. You create a handle variable the same way in which you create other variables. You can use any scope that is necessary to create the handle variable.

After you create a handle variable, you must explicitly open the handle. Then, after performing the required table I/O, you must explicitly close it.

► To use a handle

Use a Class type of handle and a data type of 7. You should name the alias for the handle the same as its table.

1. On Data Item Specifications, to complete the handle data item, click the Edit Rule tab, type the table or business view name for the handle, and then save the data dictionary item.

The data item name can be a maximum of eight characters and should be formatted as follows:

HFxxxxxx

where

HF = Designates a table I/O data item

xxxxxx = Represents the system code and group type used in the table name

For example, the table I/O data item name for table F4211 is HF4211.

You can also create a handle data item in a data structure.

2. On Event Rules Design, click the Table I/O button.
3. From Advanced Operations, click Open.
4. Click Next.
5. On Data Source, click the Handles tab.
6. Choose the handle that you want to open, and then click Next.
7. Choose a variable that contains the name of the environment in which you want to open the table.

If you want to open the table in the login environment, choose the system value SL LoginEnvironment. System values also exist for the source and target environment in Table Conversion if you use Table I/O in Table Conversion.

8. Click Finish.
9. On Data Source, click the Handles tab.
10. Choose the handle that you want to close, and then click Finish.

See Also

- ❑ *Data Dictionary* in the *Development Tools Guide* for more information about data items

Table Event Rules

You 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 called an event. When you create a PeopleSoft EnterpriseOne database trigger, you must first determine which event activates the database trigger; and then, you use Event Rules Design to create the database trigger.

Table event rules provide embedded logic at the table level. Table event rules have their own location, events, and system functions. When you use table event rules, neither the calling application nor the user is notified of changes or events to the table. No form or report interconnection is available with table event rules.

You can use table event rules for data integrity. For example, when you delete a record in Address Book, you might want to delete all associated records, such as phone and category codes. You can also use table event rules for currency. The Currency Conversion is On event rule handles currency information in table event rules.

The following is a list of the events to which you can attach event rules on a table-by-table basis:

- After Record is Deleted
- After Record is Fetched
- After Record is Inserted
- After Record is Updated
- Before Record is Deleted
- Before Record is Fetched
- Before Record is Inserted
- Before Record is Updated
- Currency Conversion is On

See Also

- ❑ *Currency* in the *Development Tools Guide* for more information about currency conversion

Creating Table Event Rules

To create a table event rule, you must do the following:

- Create the database trigger in Event Rules Design
- Generate the PeopleSoft EnterpriseOne database trigger as C code

► **To create table event rules**

1. On Object Management Workbench, check out the table to which you want to attach event rules, and then click Design.
2. On Table Design, click the Design Tools tab, and then click Start Table Trigger Design Aid.
Event Rules Design starts. You can use it to attach event rules to any of the events for the table.
3. From the Events list, choose an event.
4. Click one of the following event rule buttons:

Business Function	Attaches an existing business function.
System Function	Attaches an existing PeopleSoft EnterpriseOne system function.
If/While	Creates an IF/WHILE conditional statement.
Assign	Creates an assignment or a complex expression.
Else	Inserts an ELSE clause, which is only valid within the bounds of IF and ENDIF.
Variables	Creates a programmer-defined field, which has data dictionary characteristics for application-specific purposes but does not reside in the data dictionary.
Table I/O	Allows event rule support for database access. Performs table input and output, data validations, and record retrieval.

You do not need to create and associate data structures to the table event rule functions. The table itself is the data structure that is passed to the table event rule function.

5. On Event Rules Design, click Save to save your event rule specifications, and then click Close to return to Table Design.
6. If you are creating a new table in Table Design, click the Table Operations tab, and then click Generate Table.

Caution

Never perform this step on an existing table because it clears all data. Only perform step this for New Tables.

7. On Generate Table, complete the following fields, and then click OK:
 - Data Source
 - Password
8. On Table Design, click the Design Tools tab, and then click Build Table Triggers.

The Build Triggers option performs the following steps:

- Converts the event rule to C source code, which creates the files OBNM.c and OBNM.hxx (OBNM = Object Name). The source file will contain one function per table event rule event.
 - Creates a make file to compile the generated code.
 - Runs the make file to compile the new functions and to add them into JDBTRIG.DLL. This consolidated DLL contains table event rule functions.
9. To review a log of the build, click Generate Header File, and then open the file that is created by the system.

The creation of the table event rule is complete. The newly created or modified table event rule functions can now be called from the database APIs whenever the corresponding event occurs against the table.

Creating Dynamic Overrides

Some applications have generic form controls or grid columns in which the control properties (such as row or grid description, visual assist, and edit rules) are unknown until the user enters specific information or until specific data is loaded. In these cases, the system must dynamically override control properties at runtime. For example, an application might need to display the ending date for a field instead of static text, or you might want to change the visual assist search form when a user clicks on the visual assist button.

You can use system functions to override data dictionary properties at runtime. You can use the Set Data Dictionary Item system function to do the following:

- Override form controls and grid columns that are data dictionary items
- Change the data dictionary item completely so that all data dictionary properties are changed
- Create a new data item that is not the same type as the old item

You can use a data item name that is a string variable or a hard-coded string.

You can use the Set Data Dictionary Overrides system function for the following:

- All form controls and grid columns
- Changes to a specific data dictionary property
- Data dictionary items that are not changed

You can use the following system functions to override text at runtime:

- Set Control Text
- Set Grid Column Heading
- Set Form Title

These system functions are commonly used with text variables. You can also use the grid column system functions for Parent/Child forms. The grid column overrides work for all grid rows, and existing functionality remains intact.

You can use the following events to override a visual assist:

- Visual Assist Button Clicked
- Post Visual Assist Button Clicked

You can also use the Suppress Default Visual Assist system function to override a visual assist and affect the next visual assist form. After you suppress the default visual assist form, you can use form interconnections to call another form instead of the Search and Select form.

Working with Asynchronous Processing

A thread is a path of execution that is separate from the main path of execution. Creating a thread is similar to calling a function except that the main program continues to run, which means that two points of execution progress through the code. The different threads share the same memory space, and the operating system assigns processor time to them.

Threads allow you to process selected PeopleSoft EnterpriseOne software events or business functions in the background while the user continues to interact with the application.

Thread processing provides several performance benefits, including the following:

- Improved processing for grid and edit controls
- Background processing of certain events so that the user does not have to wait for the system to finish
- Greater flexibility for event processing
- Multiple-task processing
- Continued execution of interactive tasks, such as data entry
- Faster cursor movement

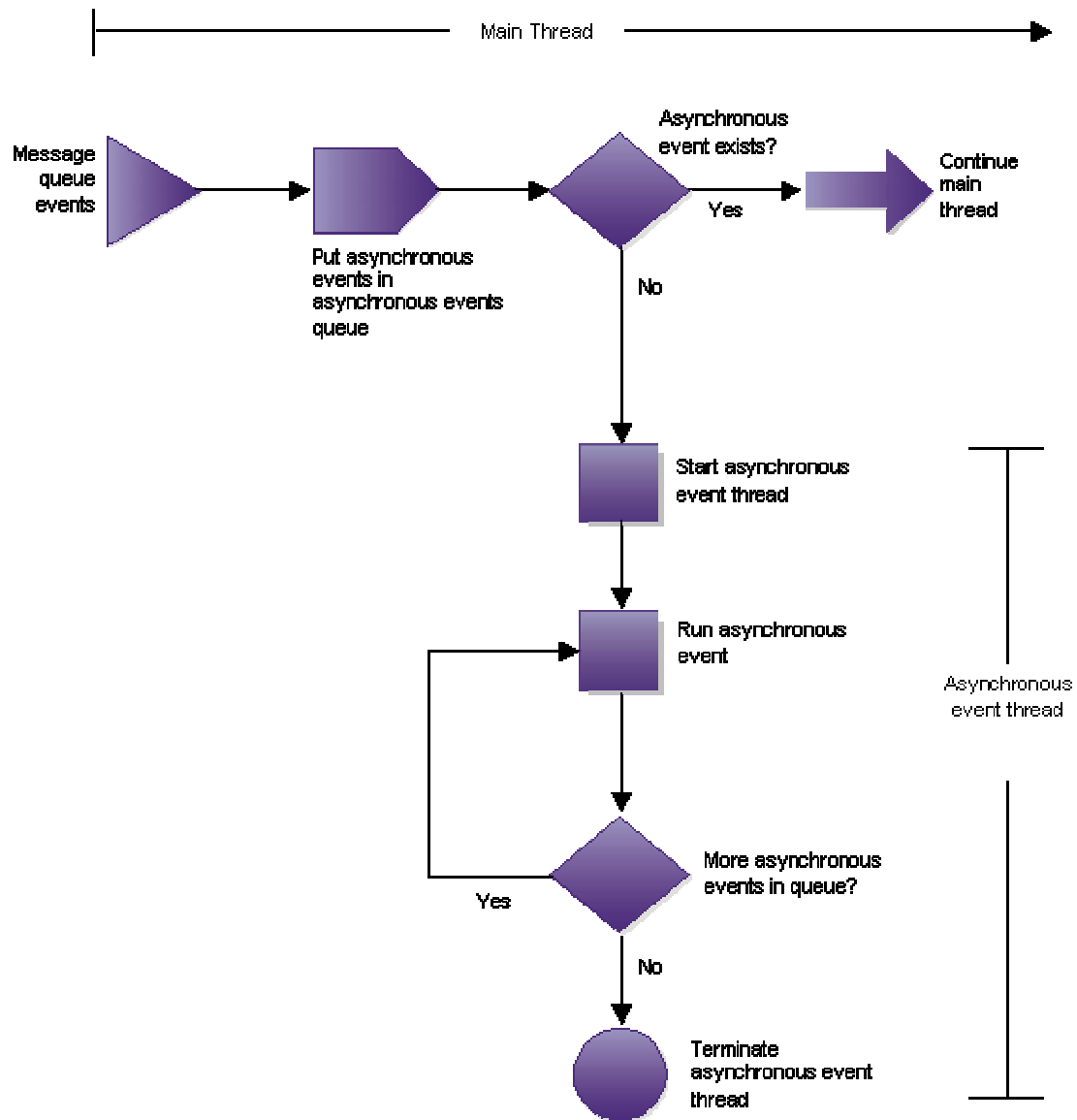
Although threads can significantly improve performance by using idle time to accomplish background processing, they also introduce the following possible problems, which are not present in single-threaded applications:

- Events processed in the main path of execution and events processed in a thread can execute at the same time. Unpredictable results can occur when two events try to access the same data at the same time.
- Asynchronous processing does not support form interconnections. You use asynchronous processes for background operations, and users should not directly interact with an asynchronous thread. For example, if you have an asynchronous process (such as Row is Exited and Changed - Asynch), you should not open a form on that worker thread because doing so requires the user to interact with a process that should be processing in the background.

Asynchronous Events

The following diagram illustrates how threaded events work.

How Threaded Events Work



Only one asynchronous event is processed at a time. Multiple instances of a unique asynchronous event might be waiting to process. The main thread continues, regardless of the presence or absence of asynchronous events in the queue. In the diagram, *Asynchronous event exists* determines whether the secondary thread starts; it does not alter the main thread.

All asynchronous events must complete their processing before OK or Cancel can be processed.

The following events can execute on a thread:

- Control is Exited and Changed - Asynch
- Row is Exited and Changed - Asynch
- Column is Exited and Changed - Asynch

Control is Exited and Changed - Asynch

As a control is exited, these three events process in the following sequence:

- Control is Exited
- Control is Exited and Changed - Inline
- Control is Exited and Changed - Asynch

The first two events are processed before the user can continue. The Control is Exited and Changed - Asynch event runs on a thread, which means that the user can continue entering data while the event runs.

Row is Exited and Changed - Asynch

As a grid row is exited, these three events process in the following sequence:

- Row is Exited
- Row is Exited and Changed - Inline
- Row is Exited and Changed - Asynch

The Row is Exited event is available on all grids. The other two events are available only on update grids (header and headerless detail forms).

The first two events are processed before the user can continue. The Row is Exited and Changed - Asynch event runs on a thread which means that the user can continue entering data while the event runs. An hourglass appears in the row header to indicate that this event is processing for a specific row.

Column is Exited and Changed - Asynch

As a grid column is exited, these three events process in the following sequence:

- Column is Exited
- Column is Exited and Changed - Inline
- Column is Exited and Changed - Asynch

These events are valid for update grids only (header and headerless detail forms).

The first two events are processed before the user can continue. The Column is Exited and Changed - Async events runs on a thread, which means the user can continue entering data while the event processes.

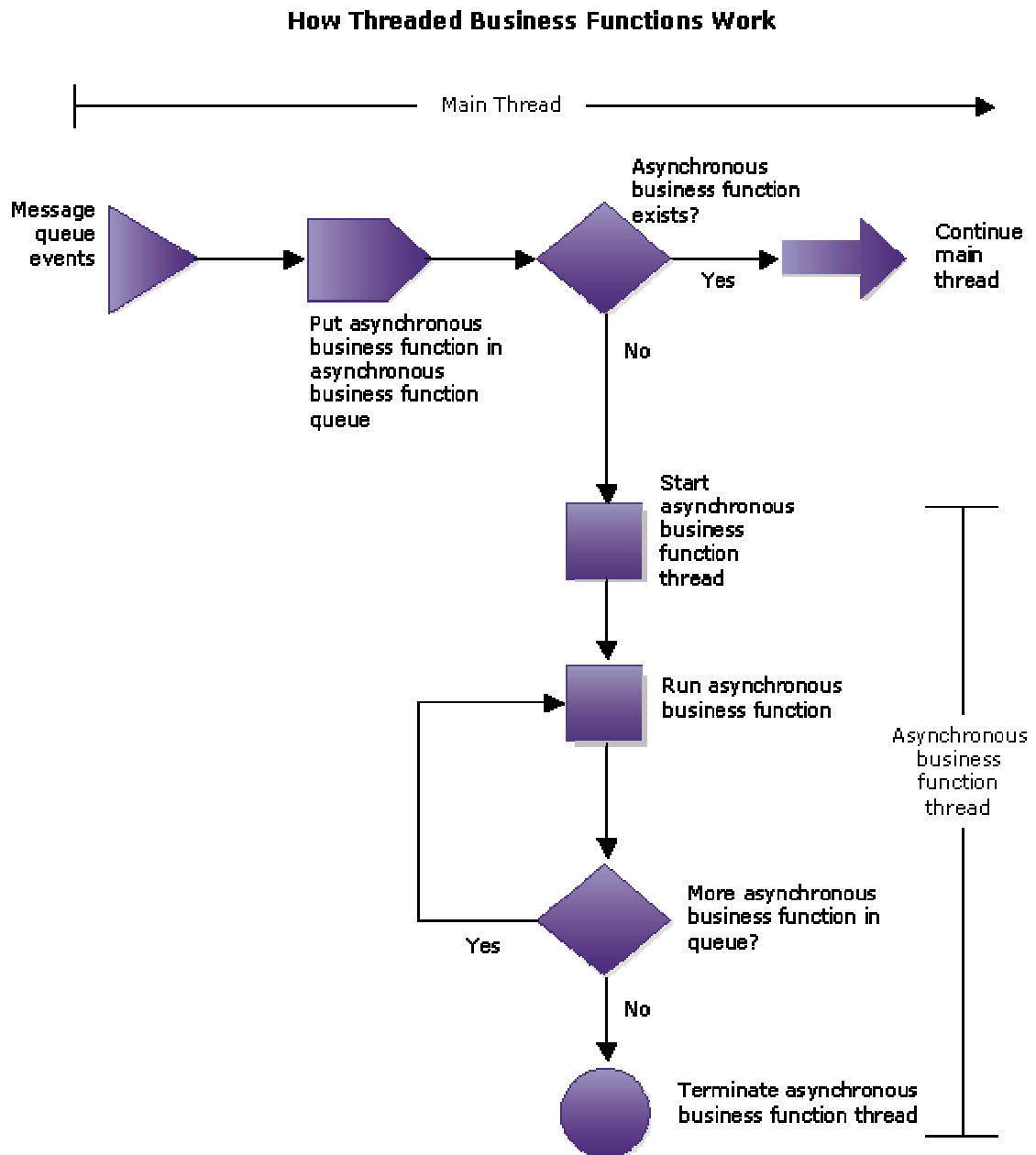
Asynchronous Business Functions

Business functions that run on OK and Cancel processing often negatively affect performance. When possible, run these business functions on a thread. Event Rules Design has an asynchronous option on the form on which business function parameters are selected. The asynchronous option is disabled for all events except OK Post Button Clicked, Cancel Post Button Clicked, and Close Post Button Clicked. When the option is turned on, the business function processes on a thread. When the option is turned on, parameters can be passed in or not passed at all. You cannot pass parameters out of the business function because the form might have reached the End Dialog event and, therefore, not open.

Attempts to set errors during asynchronous business functions are ignored. The asynchronous business function has no connection to the form, so errors cannot be set. The runtime engine can, however, use the return code from `jdeCallObject` to determine if an asynchronous business function fails. A message box appears if a failure occurs. The message box allows you to quit processing when a failure occurs. You can review the logs for more information about the failure.

Asynchronous business functions are useful for forms that make many database calls as the form is closing. For example, a good candidate for threaded business functions is a form that uses `jdeCache` calls or workfiles to store records temporarily and then writes the temporary records to the actual files when OK is clicked.

The following diagram illustrates how threaded business functions work:



Only one asynchronous business function processes at a time. The main thread continues regardless of the presence or absence of asynchronous events in the queue. In the diagram, “Asynchronous business function thread exists?” determines whether the secondary thread starts; it does not affect the main thread.

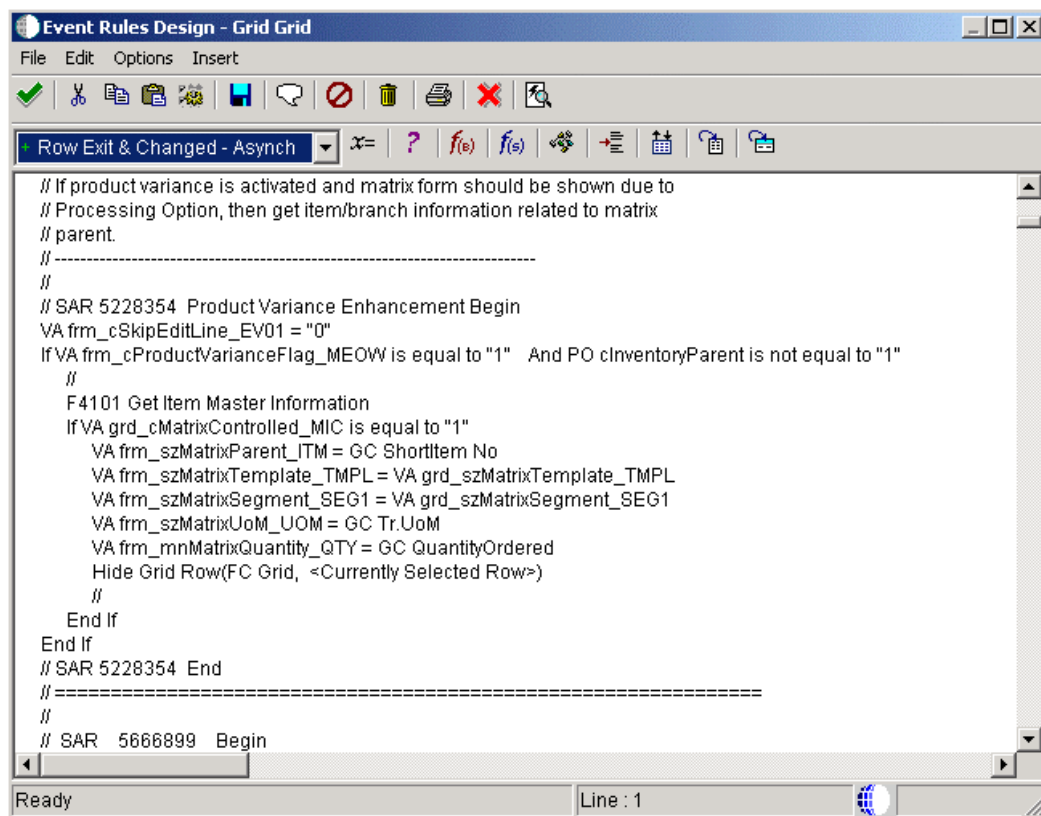
Asynchronous business functions must complete their processing before EnterpriseOne Explorer can be closed. If asynchronous business functions are still running, a message appears and EnterpriseOne Explorer cannot close.

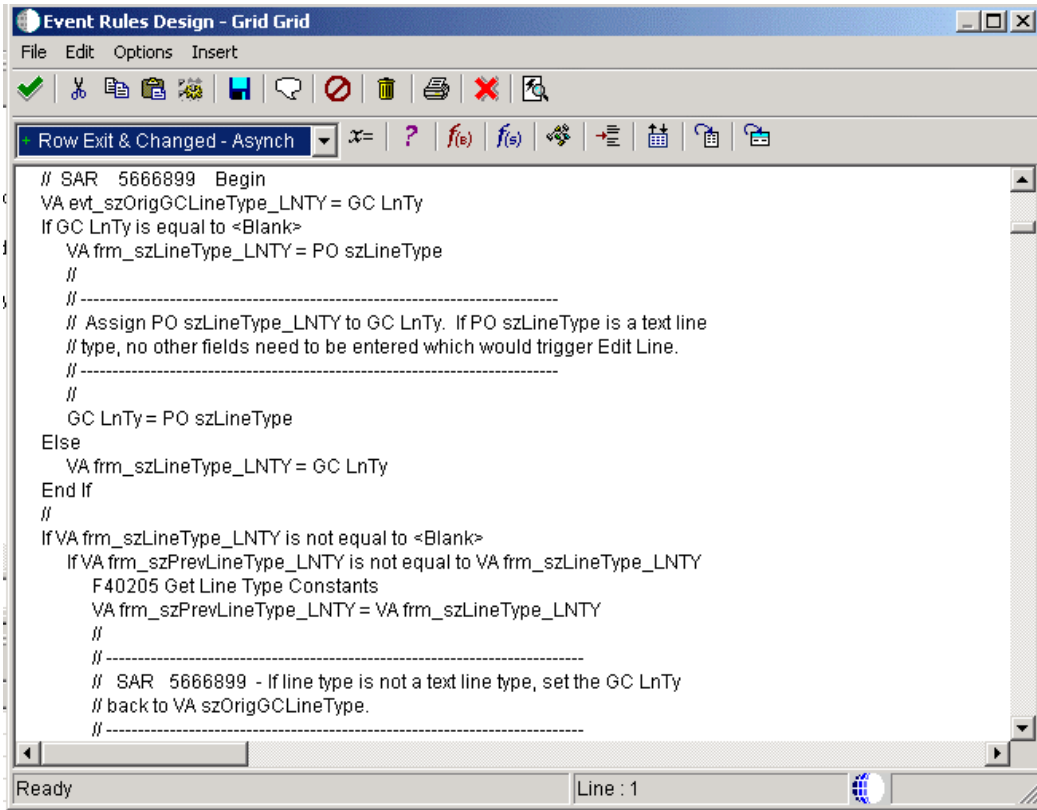
Example: Asynchronous Event Processing

The following two screens are an example of functions that are attached to the Row is exited and Changed - Asynch event.

The Row is exited and Changed - Asynch event behind the grid in the Purchase Order Detail form (P4310) has logic attached to it as follows.

The logic shown checks to see if the Product Variance is activated. If it is, then information is retrieved from the Item Master and loaded into variables for processing.





BrowsER

You can use BrowsER to view event rules and design layout for interactive and batch applications. You can enable or disable one or more event rules without extensive work in the design tools. You use this feature to debug specific event rules.

BrowsER displays the structure of forms within an interactive application, or sections within a batch application. The forms or sections appear in a hierarchical structure, with events and event rules for each section.

Working with BrowsER

You can use BrowsER to disable or enable event rules, and then observe the effect of your actions on your application.

► To work with BrowsER

1. On Object Management Workbench, select a project and then click Design.
2. On Interactive Design, click the Design Tools tab, and then click Browse Event Rules.
3. On the Browsing form, click the plus (+) and minus (-) buttons to expand or collapse the hierarchical view of events for interactive forms or batch report sections.

Each event rule appears beneath the event with which it is associated and beside a control that contains event rule logic. If it does not appear beside a control, then no event rule logic exists on that control.

4. To disable an event rule for debugging, double-click the event rule.
5. Double-click a disabled event rule to enable it.

You cannot print or modify any event rule from any BrowsER forms.

Working with BrowsER Options

You can choose one of the following BrowsER options to easily view or search for code:

- Expand Tree
- Expand Node
- Show Object IDs
- Hide Objects with no ER
- Filter ER Records
- Search

► To work with BrowsER options

On the Browsing form, right-click and choose an option from the menu that appears.

Expand Node

Expand Node allows you to expand a highlighted node to show each line of code for the node.

Show Object IDs

You use Show Object IDs for C coding. This option displays a unique ID for each object, which is useful for identifying objects that produce errors during the code generation process.

Filter ER Records

Filter ER Records allows you to search event rule records for certain types of code, including the following:

- Assignments
- Business functions
- Criterion
- Comments
- Form interconnects
- Options
- System functions

Search

Search allows you to search for a particular line of code. You can use this feature to find occurrences of particular strings or variables.

Using Visual ER Compare

Visual ER Compare is a program that lets you compare ER on your local workstation to ER in the central objects data source of any defined pathcode, TAM specifications, or ESU backup. For example, if you change the ER for an application and then want to compare your changes to the ER in the server application, you can use Visual ER Compare.

Visual ER Compare provides a line-by-line, on-screen comparison. You can change the target ER (your local version) within the utility by moving lines directly from the source ER. You can also remove or disable lines. In addition to providing an on-screen comparison, you can also print a report that lists the changes.

Launching Visual ER Compare

Apply the following task only to objects with attachable ER (applications, UBEs, tables, and business functions).

► To launch Visual ER Compare

From the Cross Application Development Tools menu (GH902), choose Object Management Workbench.

1. On Object Management Workbench, check out an object.
2. Select the object that you checked out, and then click the Design button in the center column.
3. On the Design form, click the Design Tools tab.
4. Click Visual ER Merge.
5. On Select the Location of Source Specifications, click one of the following options:
 - Central Objects Path Code
 - Remote Specifications Location
 - Software Update Backup

Understanding the Visual ER Compare Interface

When you launch Visual ER Compare, the Visual ER Compare form appears. A tree-structured menu of the ER appears on the left. The rest of the form displays the source ER (in the middle panel) and the target ER (in the right panel). The target ER is your local ER.

In the ER menu, the system uses fonts of different colors to show the rules that exist in both source and target but that are different, and the rules that exist in one location but not the other. The system indicates a change in the parent node if one or more of its children changed. In the ER panes, the system highlights the lines that differ. If lines have been added to or deleted from one side, blank lines appear on the other side. An exclamation point indicates that a line is disabled. The system uses fonts of different colors to show the rules that have changed in content or that have been added or deleted. You can change the display colors by choosing User Options from the View menu, and then choosing Set Colors.

Visual ER Compare uses an algorithm to compare lines to determine if a change has occurred. If a certain percentage of the target line is different from the source line, then the system marks the line as being different. You can change the sensitivity of the comparison by choosing User Options from the

View menu and then choosing Comparison Factors. To include disabled ER lines in the comparison, click Disable Partial Matching for Disabled ER. To change the percentage of difference required to indicate that a line is changed, enter a number in the Partial Match Ratio field. The default value of .50 indicates that a minimum of 50% of the target line must vary from the source line to trigger the system to mark it as changed.

You can choose a different source by choosing Open Source from the File menu.

Working with Visual ER Compare

Use the ER menu to identify and display specific ER components that have changed. If a parent node is identified as changed, expand it to see which of its children are different. Double-click an event in the ER menu to display its associated code. You can display more than one event at a time. Use the Tile option in the Window menu to view different ER events simultaneously.

You can also move from change to change by right-clicking in either the source or target pane and choosing either Next ER Difference (to move forward) or Previous ER Difference (to move backward).

You can change your target ER with Visual ER Compare. You can also print the changes. In addition, you can copy all of the changes from the source to the target by using the AutoMerge feature.

Changing Your Target ER

Perform any of the following actions to change your target ER:

Copy selected lines from source to target	Choose the lines to copy, right-click in the source pane, and then choose Copy Right.
Delete selected lines from the target	Choose the lines to delete, right-click in the target pane, and then choose Delete.
Enable or disable selected lines in the target	Choose the lines to enable or disable, right-click in the target pane, and then choose Enable/Disable ER.

Note

Use the shift key to select multiple contiguous lines and the control key to select multiple noncontiguous lines.

After making your changes, right-click and choose Save ER. This action saves your changes to a buffer. When you open a new source or exit Visual ER Compare, the system prompts you to save your changes again. If you elect to save your changes at this time, then the system updates the object on your workstation; otherwise, your changes are lost.

Printing a Visual ER Compare Report

You can print a report that compares the source and target ER. You can print the comparison for a particular event or for all of the ER in the object.

To print a report for an event, double-click the event in the ER menu, right-click in either pane, and then choose Print ER.

To print a report for the entire object, choose Print ER from the File menu.

Using AutoMerge

Use AutoMerge when you want the system to change the target ER to match the source ER. You can use AutoMerge to change a particular event or to update all of the ER in the object.

Caution

Before performing an AutoMerge for an entire object, compare the source and the target to be certain that you want to make all of the changes that the system detects.

To use AutoMerge on an event, double-click the event in the ER menu, right-click in either pane, and then choose AutoMerge.

To use AutoMerge on an entire object, choose Advanced Operations from the View menu, and then choose Auto Merge.

Business Functions

This topic discusses both C business functions and named event rules, and includes information about master business functions, Business Function Builder, and business function documentation.

Understanding Business Functions

You can use business functions to enhance PeopleSoft EnterpriseOne applications by grouping related business logic. Journal Entry Transactions, Calculating Depreciation, and Sales Order Transactions are examples of business functions.

You can create business functions using one of the following methods:

- Event rules scripting language

The business functions that you create using the event rules scripting language are referred to as Business Function Event Rules (also called Named Event Rules). If possible, use Business Function Event Rules for your business functions. In some instances, C business functions might better suit your needs.

- C programming code

PeopleSoft EnterpriseOne software creates a shell into which the user inserts logic using C. You use C business functions mainly for caching, but they can also be used for the following:

- Batch error level messaging
- Large functions

C business functions work better for large functions (as determined by the group). If you have a large function, you can break the code up into smaller individual functions and call them from the larger function.

- Functions for which performance is critical
- Complex Select statements

After you create business functions, you can attach them to PeopleSoft EnterpriseOne applications to provide additional power, flexibility, and control.

Components of a Business Function

The process of creating a business function produces several components. The Object Management Workbench is the entry point for the tools that create the components. The following components are created:

Component	Where Created
Business Function Specifications	Object Management Workbench Business Function Source Librarian and Business Function Design
Data Structure Specifications	Object Management Workbench Business Function Parameter Design
.C file	Generated in Business Function Design Modified with the IDE
.H file	Generated in Business Function Design Modified with the IDE

The DLLs are divided into categories. This distribution provides better separation between the major functional groups, such as tools, financials, manufacturing, distribution, and so on. Most business functions are organized into a consolidated DLL based on their system code. For example, a financials business function with system code 01 belongs in CFIN.DLL.

You should follow these guidelines when you add or modify business functions:

- Create a custom parent DLL unless you are adding a PeopleSoft EnterpriseOne business function. Assign a parent DLL to the business functions based on the system code defined in UDC table H92/PL. If no DLL is assigned for the system code in which the business function is created, use CCUSTOM, where CUSTOM is the 7-character version of the company name. You can change the DLL after the business function is created.
- When you write business function code, ensure that all calls to other business functions use the jdeCallObject protocol. Linker errors might occur if you do not use jdeCallObject and you attempt to call a business function in a different DLL. A linker error prevents your function call from working.

The following table lists some of the DLLs for which Business Function Builder manages the builds:

DLL Name	Functional Group
CAEC	Architecture
CALLBSFN	Consolidate BSFN Library
CBUSPART	Business Partner
CCONVERT	Conversion Business Functions
CCORE	Core Business Functions

CCRIN	Cross Industry Application
CDBASE	Tools - Database
CDDICT	Tools - Data Dictionary
CDESIGN	Design Business Functions
CDIST	Distribution
CFIN	Financials
CHRM	Human Resources
CINSTALL	Tools Install
CINV	Inventory
CLOC	Localization
CLOG	Logistics Functions
CMFG	Manufacturing
CMFG1	Manufacturing - Modified BFs
CMFGBASE	Manufacturing Base Functions
COBJLIB	Tools - Object Librarian
COBLIB	Busbuild Functions
COPBASE	Distribution/Logistic Base Functions
CRES	Resource Scheduling
CRUNTIME	Tools - Run Time
CSALES	Sales Order
CTOOL	Tools - Design Tools
CTRAN	Transportation
CTRANS	Tools - Translations
CWARE	Warehouse
CWRKFLOW	Tools - Workflow
JDBTRG1	Table Trigger Library 1

JDBTRG2	Table Trigger Library 2
JDBTRG3	Table Trigger Library 3
JDBTRG4	Table Trigger Library 4
JDBTRIG	Parent DLL for Database Triggers

Note

Do not use table triggers for regular business functions.

How Distributed Business Functions Work

The Object Management Workbench manages the following three main components that make up named event rules or business functions:

- Object Name
The Object Name is the actual source file.
- Function Name
The name of the business function or event rule.

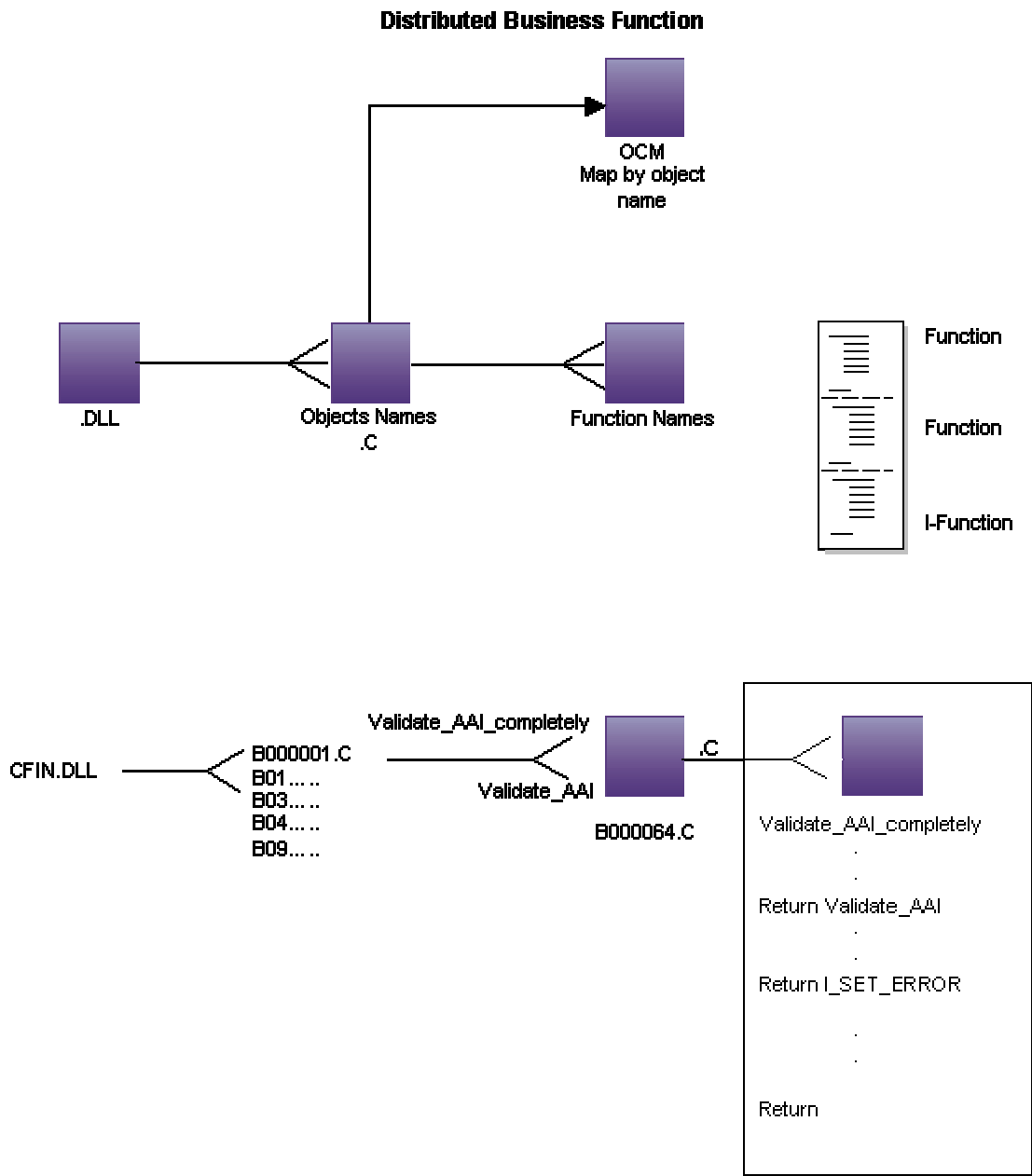
Note

Each function must have a data structure.

- DLL Name
The DLL is a dynamic link library.

When a business function is called, the Object Configuration Manager (OCM) determines where to run the business function. After the system maps a business function to a server, calls from that business function cannot be mapped back to the workstation.

The following graphic illustrates how distributed business functions work:



See Also

- ❑ *Object Configuration Manager* in the *Configurable Network Computing Implementation Guide* for more information about Object Configuration Manager

Creating Business Function Event Rules

A business function event rule (BSFN ER) is a business function object for which the source language is event rules instead of C. You create a business function event rule using the event rules scripting language. This scripting language is platform-independent and is stored in a database as a PeopleSoft EnterpriseOne software object. Business function event rules are also called Named Event Rules (NER).

Prerequisite

- ❑ Create a data structure. See *Data Structures* in the *Development Tools Guide* for more information about creating data structures.

► To design a business function event rule

1. On Object Management Workbench, create a business function.
2. On Business Function Design, click the Design Tools tab, and then choose Start Business Function Design Aid.
3. Complete the Parent DLL field.
4. Complete the Business Function Location (C/S) field to specify the business function location in which you want this business function to reside.

For Function Location, you typically choose Client/Server Function. Use the Client Only and Server Only function locations only if absolutely necessary.

5. From the Row menu, choose Parameters.
6. On Business Function Parameters, choose Select from the Form menu.
7. Click Find to locate the data structure to which the business function should be attached.

You might need to create a new data structure if an existing one does not meet your needs.

8. To create a new data structure, click Add and complete the steps for creating data structures.

If the business function already has a data structure, the system identifies the business structure. You can choose another data structure or edit the one that you want to use.

► To create or edit event rules for a business function event rule

1. On Business Function Design, choose Edit from the Form menu to create or edit the event rules that comprise the function.

When you create a new BSFN event rule, ensure that you choose NER as the Source Language.

2. From the File menu, choose Exit.
3. On Business Function Event Rules Design, construct the business function event rule.
4. Click the Save button or the OK button to save the event rules source code and return to Business Function Design.
5. On Business Function Design, click OK to save the functions and return to Business Function Source Librarian.

6. On the Design Tools tab, choose Build Business Function to create the .c and .h files, create a make file, and build the business function.

Because the source language is NER, Business Function Source Librarian creates an objname.c and objname.h file for the business function event rule. The business function event rule resides in the parent DLL that is associated with the business function.

7. To attach the newly created business function event rule to an event, click the *f(B)* button on Event Rules Design.

Ensure that the source language is NER when you browse available business function objects.

Example: Named Event Rule

Named event rules are modular. That is, they can be reused in multiple places by multiple programs. This modularity reduces rework and allows you to reuse code.

Not all chunks of code should be packaged in a business function module. For example, when code is so specific that it applies only to a particular program, and it is not reused by any other programs, you should leave it in one place instead of packaging it in a business function. You can attach all the logic on a hidden control (*Button Clicked* event) and use a system function to process the logic as needed.

An example of a named event rule is N3201030. This business function creates generic text and Work Order detail records (F4802) for a configured work order. Based on the structure of the sales order in F3296, the configured segments for the item on the passed work order and all lower level segments are included in the generic text.

The following example illustrates the function as it appears in Event Rules Design:

Named Event Rule Begin

```
//
// Convert the related sales order number into a math numeric. If
that fails
// exit the function
//
String, Convert String to Numeric
If VA evt_cErrorCode is equal to "1"
//
// Validate that the work order item is a configured item.
//
F4102 Get Item Manufacturing Information
If VA evt_cStockingType is not equal to "C"
    And BF cSuppressErrorMessages is not equal to "1"
BF szErrorMessageID = "3743"
Else
BF szErrorMessageID = " "
//
// Delete all existing "A" records from F4802 for this work order.
//
VA evt_cWODetailRecordType = "A"
F4802.Delete
F4802.Close
//
// Get the segment delimiter from configurator constants.
//
F3293 Get Configurator Constant Row
If VA evt_cSegmentDelimiter is less than or equal to <Blank>
```

```

    VA evt_cSegmentDelimiter - "/"
End If
//

F3296.Open
F3296.Select
If SV File_IO_Status is equal to CO
SUCCESS

F3296.FetchNext
//
// Retrieve the F3296 record of the work order item. and determine
its key
// sequence by parsing ATSQ looking for the last occurrence of "1".
The substring
// of ATSQ to this point becomes the key for finding the lower level
configured
// strings
//
If VA evt_mnCurrentSOLine is equal to BF mnRelatedSalesOrderLineNumber
// Get the corresponding record from F32943. Process the results of
that fetch
// through B3200600 to add the parent work order configuration to the
work order
// generic text.
F32943.FetchSingle
If SV File_IO_Status is equal to CO SUCCESS

    VA evt_szConfiguredString = concat([VA
    evt_ConfiguredStringSegment01],
    [VA evt_ConfiguredStringSegment02])
    Config String Format Segments Cache

End If
//
// Find the last level in ATSQ that is not "00". Note that the first
three
// characters represent the SO Line Number to the left of the decimal.
Example:
// SO Line 13.001 will have the ATSQ characters "013". Each configured
item can have
// 99 lower-level P-Rule items and a total of ten levels. Therefore
every pair
// thereafter is tested.
//
VA evt_mnSequencePosition - "1"
While VA evt_mnSequencePosition is less than "23"
And VA evt_szCharacterPair is not equal to "00"
VA evt_mnSequencePosition - [VA evt_mnSequencePosition] + 2
VA evt_szCharacterPair = substr([VA evt_szTempATSQ],[VA
evt_mnSequencePosition],2)
End While
VA evt_szParentATSQ = substr([VA evt_szTempATSQ],0,[VA
evt_mnSequencePosition])
//
// For each record in F3296 for the related sales order, find those
with the same

```

```

// key substring of ATSQ. Retrieve the associated record from F32943
if
// available and pass the configured string to N3200600 for addition
to the work
// order generic text.
//
F3296.FetchNext
While SV File_IO_Status                      is equal to CO SUCCESS

VA evt_szChildATSQ = substr([VA evt_szTempATSQ],0,[VA
evt_mnSequencePosition])
If VA evt_szChildATSQ is equal to VA evt_szParentATSQ
F32943.FetchSingle
If SV File_IO_Status                      is equal to CO SUCCESS

VA evt_szConfiguredString = concat([VA evt_ConfiguredStringSegment01],
[VA evt_ConfiguredStringSegment02])
Config String Format Segments Cache
End If
End If
F3296.FetchNext
End While
F32943.Close
//
// Unload segments cache into the work order generic text. B3200600
Mode 6
Config String Format Segments Cache
//
End If
End If
F3296.Close
//
End If
Else
// The related sales order number is invalid. Return an error.
If BF cSuppressErrorMessages is not equal to "1"
Set NER Error ("0002", BF SzRelatedSalesOrderNumber)
End If
End If

```

Named Event Rule End

C Business Functions

PeopleSoft EnterpriseOne software contains two types of business functions: Named Event Rules (NER) and C business functions. C business functions are written in C programming language and are used to perform functions that are not available in NER. C business functions include both a header file (.h) and a source file (.c).

Understanding Header File Sections

The following table describes the major sections of a business function header file:

Section	What It Includes	Description and Guidelines
Header File Comment	- Header file name - Description - History - Programmer - Software action request (SAR) number - Copyright information	Comments that the input process of the Business Function Source Librarian builds. The programmer name and SAR number are manually updated by the programmer.
Table Header Inclusions	Include statements for header files associated with tables that are directly accessed by this business function	Table header files include definitions for the fields in a table and the ID of the table itself.
External Business Function Header Inclusions	Include statements for headers associated with externally defined business functions that are directly accessed by this business function	External function calls with jdeCallObject are included to use the predefined data structures.
Global Definitions	Global constants used by the business function	Symbolic names that you enter in uppercase; words are separated by an underscore character. Use global definitions sparingly.
Structure Type Definitions	Data structure definitions for internal processing	To prevent naming conflicts, define this structure using structure names that are prefixed by the source file name.
DS Template Type Definition	- Data structure type definitions generated by Business Function Design - Symbolic constants for the data structure generated by Business Function Design	This structure must be modified through the Object Management Workbench.
Source Preprocessor	- Undefines JDEBFRTN if it is already defined - Checks for how to define JDEBFRTN - Defines JDEBFRTN	Ensures that the business function declaration and prototype are properly defined for the environment and source file, including this header.
Business Function Prototype	Prototypes for all business functions in the source file	Defines the business functions in the source file, the parameters that are passed to them, and the type of value that they return.
Internal Function Prototype	Prototypes for all internal functions that are required to support business functions within this source file	Defines the internal functions that are associated with the business functions in the source file, the parameters that are passed to each internal function, and the type of value that they return.

Example: Business Function Header File

Assume that Business Function Design created the following header file. This file contains only the required components in a business function header file.

Header File Begin

```

/*****
*****
*   Header File:   B99TEST.h
*
*   Description:   test Header File
*
*   History:
*       Date           Programmer   SAR# - Description
*       -----
-----
*   Author 10/14/2003   DEMO           Unknown - Created
*
*
*   Copyright (c) 1994   PeopleSoft, Inc., 2003
*
*   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 __B99TEST_H
#define __B99TEST_H
/*****
*****
*   Table Header Inclusions

*****/

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

*****/

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

*****/

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

```

```

* Structure Definitions

*****/

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

*****/
/*****
* TYPEDEF for Data Structure
*   Template Name: Test Data Structure
*   Template ID:   D59TEST
*   Generated:     Tue Oct 14 16:53:08 2003
*
* DO NOT EDIT THE FOLLOWING TYPEDEF
*   To make modifications, use the EnterpriseOne Data Structure
*   Tool to Generate a revised version, and paste from
*   the clipboard.
*
*****/

#ifndef DATASTRUCTURE_D59TEST
#define DATASTRUCTURE_D59TEST

typedef struct tagDSD59TEST
{
    JCHAR          cEverestEventPoint01;
    JCHAR          szNameAlpha[41];
    MATH_NUMERIC   mnAmountField;
} DSD59TEST, *LPDSD59TEST;

#define IDERRcEverestEventPoint01_1    1L
#define IDERRszNameAlpha_2            2L
#define IDERRmnAmountField_3          3L

#endif

/*****
*****
* 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 F0101Test
    (LPBHVRCOM lpBhvrCom, LPVOID lpVoid, LPDSD0100018 lpDS);

/*****
* Internal Function Prototypes

*****/

#endif    /* __B99TEST_H */

```

Header File End

The following table describes the contents of the various lines in the header file:

	Header File Line	Where Input	Description
1.	Header File	Object Management Workbench	Verify the name of the business function header file.
2.	Description	Object Management Workbench	Verify the description.
3.	History	IDE (Integrated Development Environment)	Manually update the modification log with the programmer name and the appropriate SAR number.
4.	#ifndef	Business Function Design	Symbolic constant prevents the contents from being included multiple times.
5.	Table Header Inclusion	Business Function Design	When business functions access tables, related tables are input and Business Function Design generates an include statement for the table header file.
6.	External Business Function Header Inclusions	Business Function Design	No external business functions for this application.
7.	Global Definitions	IDE (Integrated Development Environment)	Constants and definitions for the business function. PeopleSoft does not recommend using this block. Global variables are not recommended. Global definitions go in .c not .h.
8.	Structure Definitions	IDE (Integrated Development Environment)	Data structures for passing information between business functions, internal functions, and database APIs.

	Header File Line	Where Input	Description
9.	TYPEDEF for Data Structure	Business Function Design	Data structure type definition. Used to pass information between an application or report and a business function. The programmer places it on the Clipboard and pastes it in the header file. Its components include: • Comment Block, which describes the data structure • Preprocessor Directives, which ensure that the data type is defined only once • Typedef, which defines the new data type • #define, which contains the ID to be used in processing if the related data structure element is in error • #endif, which ends the definition of the data structure type definition and its related information
10.	Source Preprocessor Definitions	Business Function Design	All business function header files contain this section to ensure that the business function is prototyped and declared based on where this header is included.
11.	Business Function Prototype	Business Function Design	Used for prototypes of the business function.
12.	JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode	Business Function Design	Business Function Standard All business functions share the same return type and parameter data types. Only the function name and the data structure number vary between business functions. Parameters include: • LPBHVRCOM: Pointer to a data structure used for communicating with business functions. Values include an environment handle. • LPVOID: Pointer to a void data structure. Currently used for error processing; will be used for security in the future. • LPDS#####: Pointer to a data structure containing information that is passed between the business function and the application or report that invoked it. This number is generated through Object Librarian. • Parameter names (lpBhvrCom, lpVoid, and lpDS) will be the same for all business functions. • JDEBFRTN(ID)JDEBFWINAPI: All business functions will be declared with this return type. It ensures that they are exported and imported properly.

	Header File Line	Where Input	Description
13.	Internal Function Prototypes	Business Function Design	Internal function prototypes required to support the business functions in this source file.

Understanding the Structure of a Business Function Source File

The Object Management Workbench builds a template for the business function source file. The business function source file consists of several major sections, as described in the following table:

Section	What It Includes	Description
Source File Comment Block	- Source file name - Description - History - Programmer - Date - SAR Number - Description - Copyright information	Built from the information in the Business Function Source Librarian. The programmer manually updates the programmer name and SAR number.
Notes Comment Block	Any additional relevant notes concerning the business function source	Document complex algorithms used, how the business functions in the source relate to each other, and so on.
Business Function Comment Block	- Business function name - Description - Description list of the parameters	
Business Function Source Code	Source code for the business function	
Internal Function Comment Block	- Function name - Notes - Returns - Parameters	Copy these blocks and place the values in the specified sections to describe the internal function. Follow the comment block with internal function source code.
Internal Function Source Code	Source code for the internal function described in the comment block	The business function developer enters this code as needed. A populated internal function comment block must precede this code.

Example: Business Function Source File

Assume that Business Function Design created the following source file called Check for In Add Mode. It contains the minimum components required in a business function source file. The source code in the Main Processing section is entered manually, and varies from business function to business function. All other components are generated by Business Function Design.

Source File Begin

```
#include <jde.h>
#define b999test_c

/*****
 *
 * Source File:  b99test
 *
 * Description:  test Source File
 *
 * History:
 *      Date          Programmer  SAR# - Description
 *      -----
 *
 * Author 10/14/2003  DEMO          Unknown - Created
 *
 * Copyright (c) 1994  PeopleSoft, Inc.
 *
 * 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 <b99test.h>

/*****
 *
 * Business Function:  F0101Test
 *
 * Description:  F0101Test
 *
 * Parameters:
 *      LPBHVRCOM          lpBhvrCom      Business Function
Communications
 *      LPVOID             lpVoid         Void Parameter - DO NOT USE!
 *      LPDSD010           lpDS           Parameter Data Structure
Pointer
 *
 *****/

JDEBFRTN(ID) JDEBFWINAPI F0101Test (LPBHVRCOM lpBhvrCom, LPVOID
lpVoid,
LPDSD0100018 lpDS)
{
```

```

/*****
***
*   Variable declarations
***
****/

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

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

/*****
***
*   Check for NULL pointers
***
****/
    if ((lpBhvrCom == (LPBHVRCOM) NULL) ||
        (lpVoid == (LPVOID) NULL) ||
        (lpDS == (PLDSD0100018) NULL))
    {
        jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, _J("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 a source file
number                               // and "a" with the function name
*
*       Notes:
*
*       Returns:
*
*   Parameters:
*****
**/

```

Source File End

The lines that appear in the source file are described in the following table:

	Source File Line	Where Input	Description and Guidelines
1.	#include <jde.h>	Business Function Design	Includes all base PeopleSoft EnterpriseOne definitions.
2.	#define b98sa001_c	Business Function Design	Ensures that related header file definitions are correctly created for this source file.
3.	Source File	Object Management Workbench	Verifies the information in the file comment section. Enter the programmer's name, SAR number, and description.
4.	#include <B98SA001.h>	Object Management Workbench	Includes the header file for this application.
5.	Business Function	Business Function Design	Verifies the name and description in the business function comment block.

	Source File Line	Where Input	Description and Guidelines
6.	JDEBFRTN(ID) JDEBFWINAPI CheckForInAdd Mode (LPBHVRCOM lpBhvrCom, LPVOID lpVoid, LPDS104438 lpDS)	Business Function Design	Includes the header of a business function declaration.
7.	Variable declarations	IDE	Declares variables that are local to the business function.
8.	Declare structures	IDE	Declares local data structures to communicate between business functions, internal functions, and the database.
9.	Declare pointers	IDE	Declares pointers.
10.	Check for NULL pointers	Business Function Design	Business Function Standard Verifies that all communication structures between an application and the business function are valid.
11.	jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, _J("4363"), LPVOID) NULL); return ER_ERROR;	Business Function Design	Sets the standard error to be returned to the calling application when any of the communication data structures are invalid.
12.	Set pointers	IDE	Declares and assigns appropriate values to pointers.
13.	Main Processing	IDE	Provides main functionality for a business function.
14.	Function Clean Up	IDE	Frees any dynamically allocated memory.

	Source File Line	Where Input	Description and Guidelines
15.	Internal function comment block	IDE	Defines internal functions that are required to support the business function. They should follow the same C coding standards. A comment block is required for each internal function and should be formatted as shown in the example.

Use the MATH_NUMERIC data type exclusively to represent all numeric values in PeopleSoft EnterpriseOne software. The values of all numeric fields on a form or batch process are communicated to business functions in the form of pointers to MATH_NUMERIC data structures. MATH_NUMERIC is used as a data dictionary data type.

Using Application Programming Interfaces (APIs)

Application programming interfaces (APIs) are routines that perform predefined tasks. PeopleSoft EnterpriseOne APIs make it easier for third-party applications to interact with PeopleSoft EnterpriseOne software. These APIs are functions that you can use to manipulate PeopleSoft EnterpriseOne data types, provide common functionality, and access the database. Several categories of APIs exist, including the Common Library Routines and PeopleSoft EnterpriseOne Database (JDEBASE) APIs.

Programs using the PeopleSoft EnterpriseOne APIs are flexible for the following reasons:

- No code modifications are required as functionality is upgraded
- When a data structure changes, source modifications are minimal to nonexistent
- Common functionality is provided through the APIs, and they are less prone to error

When the code in an API changes, business functions typically have only to be recompiled and relinked.

Using Common Library APIs

The Common Library APIs, such as determining whether foreign currency is enabled, manipulating the date format, retrieving link list information, or retrieving math numeric and date information, are specific to PeopleSoft EnterpriseOne functionality. You can use these APIs to set up data by calling APIs and modifying data after API calls. Some of the more commonly used categories of APIs include MATH_NUMERIC, JDEDATE, and LINKLIST. Other miscellaneous Common Library APIs are also available.

PeopleSoft EnterpriseOne provides the following two main data types for use when creating business functions:

- MATH_NUMERIC
- JDEDATE

Because these data types might change, you must use the Common Library APIs provided by PeopleSoft EnterpriseOne to manipulate the variables of these data types.

MATH_NUMERIC Data Type

The MATH_NUMERIC data type exclusively represents all numeric values in PeopleSoft EnterpriseOne software. The values of all numeric fields on a form or batch process are communicated to business functions in the form of pointers to MATH_NUMERIC data structures. MATH_NUMERIC is used as a data dictionary data type.

The data type is defined as follows:

```
struct tagMATH_NUMERIC
{
    ZCHAR  String[MAXLEN_MATH_NUMERIC+1];/* Just the digits - no separators */
    BYTE   Sign;                        /* '-' if negative, 0x00 otherwise */
    ZCHAR  EditCode;                    /* The Data Dictionary edit code to Format for
display */
    short  nDecimalPosition;            /* # of digits from right end of string to decimal
point */
    short  nLength;                     /* The number of digits in s */
    WORD   wFlags;                      /* Processing Flags */
    ZCHAR  szCurrency[CURRENCY_CODE_SIZE];/* The Currency Code */
    short  nCurrencyDecimals;           /* The Number of Currency Decimals */
    short  nPrecision;                  /* The Data Dictionary Size */
};
```

The various elements are described in the following table:

MATH_NUMERIC Element	Description
String	Digits without separators
Sign	A minus sign indicates the number is negative, otherwise the value is 0x00
EditCode	Data dictionary edit code that formats the number for display
nDecimalPosition	Number of digits from the right to place the decimal
nLength	Number of digits in the string
wFlags	Processing flags
szCurrency	Currency code
nCurrencyDecimals	Number of currency decimals
nPrecision	Data dictionary size

JDEDATE Data Type

The JDEDATE data type exclusively represents all dates in PeopleSoft EnterpriseOne software. The values of all date fields on a form or batch process are communicated to business functions in the form of pointers to JDEDATE data structures. JDEDATE is used as a data dictionary data type.

The data type is defined as follows:

```
struct tagJDEDATE
{
    short nYear;;
    short nMonth;;
    short nDay;
};

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

The elements in the JDEDATE data type are describe in the following table:

JDEDATE Element	Description
nYear	Year (4 digits)
nMonth	Month
nDay	Day

Using Database APIs

PeopleSoft EnterpriseOne software supports multiple databases. An application can access data from a number of databases. APIs offer the following advantages:

- A standard interface to multiple database management systems
- Minimal knowledge of SQL to perform complex database operations
- Include add, modify, delete, and query operations for database systems
- Manage memory areas for passing data to and from a database
- Create and execute SQL statements at runtime
- More flexible than embedded SQL
- An improved method for developing applications in a client/server environment
- Standard return codes from function calls

Understanding Standards and Portability

The following standards affect the development of relational databases:

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

Ideally, industry standards allow users to work identically with different relational database systems. Although each major vendor supports industry standards, it also offers extensions to enhance the functionality of the SQL language. Vendors also periodically release upgrades and new versions of their products.

These extensions and upgrades affect portability. Due to the industry impact of software development, applications need a standard interface to databases that is not affected by differences between database vendors. When a vendor provides a new release, the affect on existing applications should be minimal. To solve many of these portability issues, many organizations use standard database interfaces called open database connectivity (ODBC).

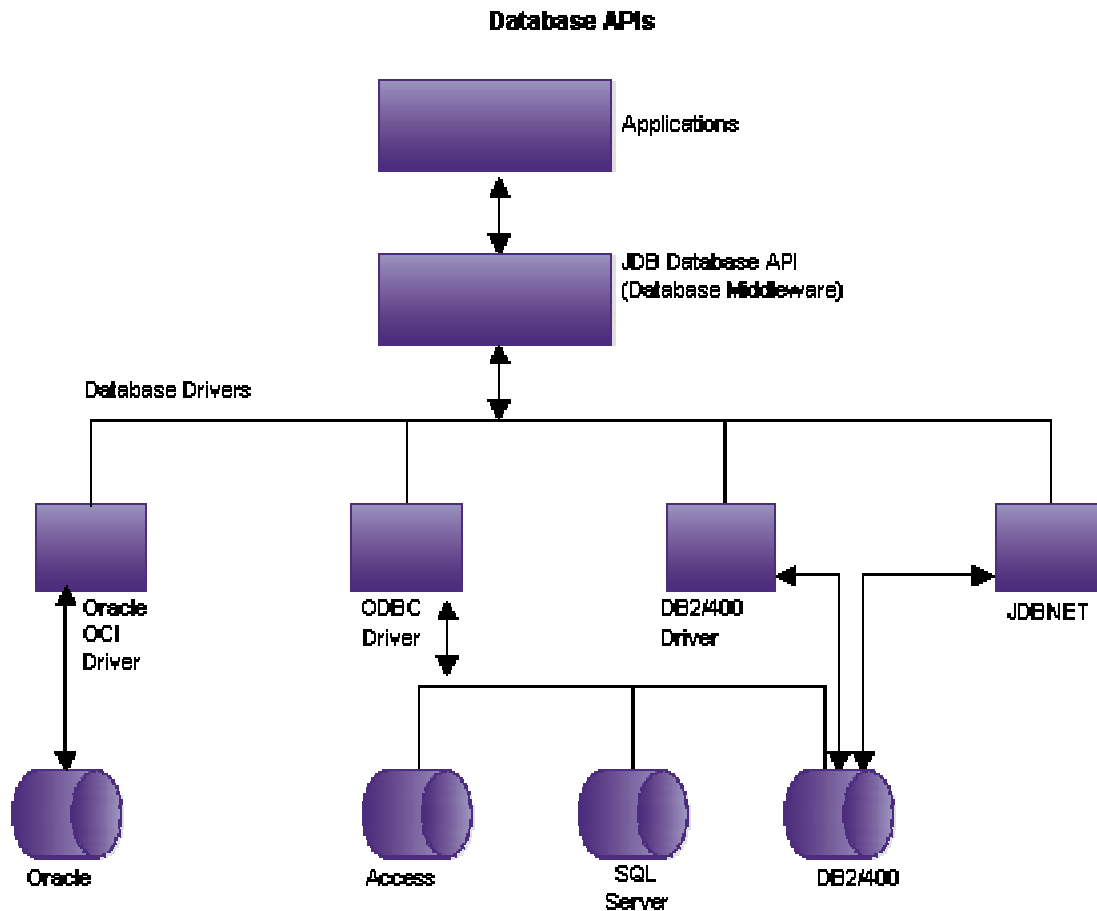
PeopleSoft EnterpriseOne Open Database Connectivity (ODBC)

PeopleSoft EnterpriseOne ODBC provides a single method to access multiple relational database management systems. ODBC allows you to use one set of functions to interface with different types of databases. You can develop and compile applications without knowing the type of database that each application uses. Database drivers are installed that allow the PeopleSoft EnterpriseOne ODBC interface to communicate with a specific database system.

A driver is an application that processes the API request, communicates with the database, and returns the result to the API. The driver also handles the input/output (I/O) buffers to the database, which allows a programmer to write an application that communicates with a generic data source. The database driver is responsible for processing the API request and communicating with the correct data source. The application does not have to be recompiled to work with other databases. If the application needs to perform the same operation with another database, a new driver is loaded.

A driver manager handles all application requests to the PeopleSoft EnterpriseOne database function call. The driver manager processes the request or passes it to an appropriate driver.

The following graphic illustrates how PeopleSoft EnterpriseOne software uses database APIs:



PeopleSoft EnterpriseOne applications access data from heterogeneous databases, using the JDB API to interface between the applications and multiple databases. Applications and business functions use the JDB API to dynamically generate platform-specific SQL statements. JDB also supports additional features, such as replication and cross-data source joins.

Standard JDEBASE API Categories

The following table describes the categories of JDEBASE APIs. You can use control and request level APIs to develop and test business functions.

Category	Description
Control Level	Provides functions for initializing and terminating the database connection.
Request Level	Provides functions for performing database transactions. The request level functions perform the following tasks: - Connect to and disconnect from tables and business views in the database - Perform data manipulation operations of select, insert, update, and

	delete Retrieve data with fetch commands
Column Level	Performs and modifies information for columns and tables.
Global Table/Column Specifications	Provides the capability to create and manipulate column specifications.

Connecting to a Database

To perform a request, the driver manager and driver must manage the information for the development environment, each application connection, and the SQL statement. The pointers that return this information to the application are called handles. The APIs must include these handles in each function call. Handles used by the development environment include the following:

Handle	Purpose
HENV	The environment handle contains information related to the current database connection and valid connection handles. Every application connecting to the database must have an environment handle. This handle is required to connect to a data source.
HUSER	The user handle contains information related to a specific connection. Each user handle has an associated environment handle with it. A connection handle is required to connect to a data source. If you are using transaction processing, initializing HUSER indicates the beginning of a transaction.
HREQUEST	The request handle contains information related to a specific request to a data source. An application must have a request handle before executing SQL statements. Each request handle is associated with a user handle.

Understanding Database Communication Steps

You can use several APIs to perform the following steps for database communication:

- Initialize communication with the database.
- Establish a connection to the specific data to access.
- Execute statements on the database.
- Release the connection to the database.
- Terminate communication with the database.

The following table describes some of the API levels, and the communication handles and API names that are associated with them.

API Level	Communication Handles	API Name
Control level (application or test driver)	Environment handle	JDB_InitEnv
Control level (application or test driver)	User handle (created)	JDB_InitUser
Request level (business function)	User handle (retrieved)	JDB_InitBhvr

Request level (business function)	Request handle	JDB_OpenTable
	Request handle	JDB_FetchKeyed()
	Request handle	JDB_CloseTable
	User handle	JDB_FreeBhvr
Control level (application or test driver)	User handle	JDB_FreeUser
Control level (application or test driver)	Environment handle	JDB_FreeEnv

Calling an API from an External Business Function

You can call APIs from external business functions. To call an API from an external business function, you must first determine the function-calling convention of the .dll that you are going to use. It can be either cdecl or stdcall. The code might change slightly depending on the calling convention. This information should be included in the documentation for the .dll. If you do not know the calling convention of the .dll, you can execute the *dumpbin* command to determine the calling convention. Execute the following command from the MSDOS prompt window:

```
dumpbin /EXPORTS ExternalDll.DLL.
```

Dumpbin displays information about the dll. If the output contains function names preceded by _ and followed by an @ sign with additional digits, the dll uses the stdcall calling convention; otherwise, it uses cdecl.

Stdcall Calling Convention

The following example is standard code for Windows programs. It is not specific to PeopleSoft EnterpriseOne software.

```
# ifdef JDENV_PC

HINSTANCE hLibrary = LoadLibrary(_TEXT("YOUR_LIBRARY.DLL")); // substitute
the name of the external dll

if(hLibrary)
{
    // create a typedef for the function pointer based on the parameters and
    return type of the function to be called. This information can be obtained
    // from the header file of the external dll. The name of the function to
    be called in the following code is "StartInstallEngine". We create a
    typedef for

    // a function pointer named PFNSTARTINSTALLENGINE. Its return type is
    BOOL. Its parameters are HUSER, LPCTSTR, LPCTSTR, LPTSTR & LPTSTR.
    // Substitute these with parameter and return types for your particular
    API.
    typedef BOOL (*PFNSTARTINSTALLENGINE) (HUSER, LPCTSTR, LPCTSTR, LPTSTR,
    LPTSTR);
```



```

// Now create a variable for your function pointer of the type you just
// created. Then make call to GetProcAddress function with the first
// parameter as the handle to the library you just loaded. The second
// parameter should be the name of the function you want to call prepended
// with an "_", and appended with an "@" followed by the total number of
// bytes for your parameters. In this example, the total number of bytes in
// the
// parameters for "StartInstallEngine is 20 ( 4 bytes for each parameter
// ). The "GetProcAddress" API will return a pointer to the function that you
// need to
// call.

PFNSTARTINSTALLENGINE lpfnStartInstallEngine = (PFNSTARTINSTALLENGINE)
GetProcAddress(hLibrary, "_StartInstallEngine@20");

if ( lpfnStartInstallEngine )
{
// Now call the API by passing in the requisite parameters.

lpfnStartInstallEngine(hUser, szObjectName, szVersionName, pszObjectText,
szObjectType);
}

#endif

```

Cdecl Calling Convention

The process for using the cdecl calling convention is similar to the process for using the std calling convention. The main difference is the second parameter for GetProcAddress. Note the comments that precede that call.

```

# ifdef JDENV_PC

HINSTANCE hLibrary = LoadLibrary(_TEXT("YOUR_LIBRARY.DLL")); // substitute
the name of the external dll

if(hLibrary)
{
// create a typedef for the function pointer based on the parameters and
// return type of the function to be called. This information can be obtained
// from the header file of the external dll. The name of the function to
// be called in the following code is "StartInstallEngine". We create a
// typedef for

// a function pointer named PFNSTARTINSTALLENGINE. Its return type is
// BOOL. Its parameters are HUSER, LPCTSTR, LPCTSTR, LPTSTR & LPTSTR.
// Substitute these with parameter and return types for your particular
// API.

typedef BOOL (*PFNSTARTINSTALLENGINE) (HUSER, LPCTSTR, LPCTSTR, LPTSTR,
LPTSTR);

```

```
// Now create a variable for your function pointer of the type you just
// created. Then make call to GetProcAddress function with the first
// parameter as the handle to the library you just loaded. The second
// parameter should be the name of the function you want to call. In this
// case it will be "StartInstallEngine" only. The "GetProcAddress" API
// will return a pointer to the function that you need to call.

PFNSTARTINSTALLENGINE lpfnStartInstallEngine = (PFNSTARTINSTALLENGINE)
GetProcAddress(hLibrary, "StartInstallEngine");

if ( lpfnStartInstallEngine )
{
    // Now call the API by passing in the requisite parameters.

    lpfnStartInstallEngine(hUser, szObjectName, szVersionName, pszObjectText,
szObjectType);
}

#endif
```

Note

These calls work only on a Windows *client* machine. LoadLibrary & GetProcAddress are Windows APIs. If the business function is compiled on a *server*, the compile will fail.

Calling a Visual Basic Program from PeopleSoft EnterpriseOne Software

You can call a Visual Basic program from a PeopleSoft EnterpriseOne business function and pass a parameter from the Visual Basic program to the PeopleSoft EnterpriseOne business function using the following process:

5. Write the Visual Basic program into a Visual Basic .dll that exports the function name of the program and returns a parameter to the PeopleSoft EnterpriseOne business function.
6. Write a business function that loads the Visual Basic .dll using the win32 function LoadLibrary.
7. In the business function that you create, call the win32 function GetProcAddress to get the Visual Basic function and call it.

Working with Business Functions

Every business function must follow a defined structure and form. Every line of code must conform to the PeopleSoft EnterpriseOne business function programming standards. Following these standards will allow you to take advantage of PeopleSoft's proven software engineering approach for developing applications. Create your business functions as follows:

- Use Object Management Workbench to build business function data structures.
- Use Object Management Workbench to create business function source and header files.

- Create and add type definitions for data structures to the header file.

See Also

- *Creating a Business Function Data Structure* in the *Development Tools Guide* for more information about creating data structures for business functions

Accessing Business Function Design

The following task describes how to access Business Function Design.

► To access Business Function Design

On Object Management Workbench, check out the business function with which you want to work.

1. Ensure that the business function is selected, and then click the Design button in the center column.
2. On Business Function Design, click the Design Tools tab, and then click Start Business Function Design Aid.

Viewing Object Codes

You use object codes to group business functions into categories. You can then use these codes as filtering criteria for searches.

► To view object codes

1. On Business Function Design, from the Row menu, choose Codes.
2. Review the following fields:
 - Source Module
 - Function Name
 - Description
 - Business Function Category
 - Function Use

Adding Related Tables and Functions

You can attach tables and functions to a business function. You must add related tables and functions to the business function object to generate the code for the source and header files.

► To add related tables

1. On Business Function Design, from the Form menu, choose Tables.
2. On Related Tables, enter the names of related tables.
3. Click OK.

► **To add related functions**

1. On Business Function Design, choose Functions from the Form menu.
2. On Related Business Functions, enter the names of related business function source objects.
3. Click OK.

The associated record is stored in the Object Librarian - Object Relationships table (F9863).

Debugging Business Functions

Because the source code for business function event rules is generated into C, you use the same procedures for debugging both C and NER business functions.

See Also

- ❑ *Debugging* in the *Development Tools Guide* for information about debugging business functions

Changing a Business Function Parent DLL or Location

You might need to change the parent DLL for a business function.

► **To change a business function parent DLL**

1. On Business Function Design, select the new DLL name in Parent DLL, or change the business function location, and then click OK.
2. Click Build Business Function to rebuild the new DLL file.
3. On Object Management Workbench - [Business Function Design], click OK.
4. On Object Management Workbench, check in the business function.

You should transfer this business function to all path codes as soon as possible.

Note

After you have changed the DLL, go to C:\B9\System\Bin32\BusBuild.exe, select the old DLL file where the business function was, and select Build from the Build menu to rebuild the file.

Your change becomes available to everyone in the next package.

Creating and Specifying Custom DLLs

Business function DLLs are consolidated. Therefore, you need to build each of your custom business functions into a custom DLL that you create. This process ensures that your custom business functions remain separate from PeopleSoft EnterpriseOne business functions. The build program reviews the Object Librarian Master Table (F9860) to verify that your custom DLL exists.

Creating a Custom DLL

Before you can build a custom business function into a custom DLL, that custom DLL must exist in the Object Librarian Master Table (F9860). Therefore, you must create a custom DLL.

► To create a custom DLL

1. On Object Management Workbench, click Add.
2. On Add J.D. Edwards Object to the Project, choose the Business Function Library option, and then click OK.
3. On Add Object, complete the following fields and click OK.
 - Object Name
 - Description
 - Product Code
 - Product System Code
 - Object Use
4. Exit EnterpriseOne.
5. In Windows, from the Start menu, select Run.
6. Type C:\B9\System\Bin32\BusBuild.exe.
7. From the Build menu, select Rebuild Libraries to rebuild all libraries.

This process takes several minutes.

Specifying a Custom DLL for a Custom Business Function

When you create a custom business function, you need to specify one of your custom DLLs. If you don't, the build process builds the custom business function into the PeopleSoft EnterpriseOne CCUSTOM.DLL, where CUSTOM is the seven-character name of the company, which is the default.

► To specify a custom DLL for a custom business function

After you create a custom DLL, you must specify that custom DLL.

1. Enter the custom DLL name in the Parent DLL field, and click OK.

Note

You can also change the business function location if necessary.

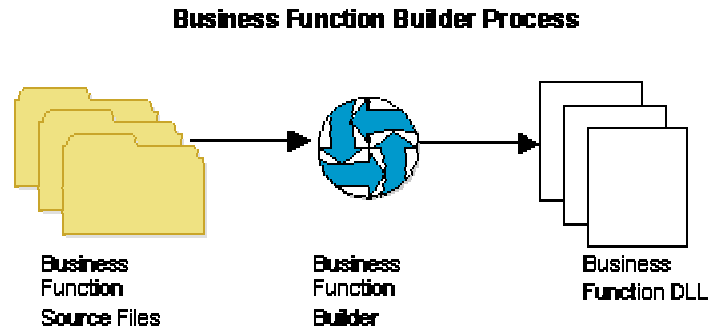
2. Run the build for the business function.

Working with Business Function Builder

You use Business Function Builder to build business function code into a dynamic link library (DLL). You can build C business functions, Named Event Rules, and table event rules. The process that occurs when you run Business Function Builder to build business functions includes compiling and

linking. Compiling involves creating a business function object. Linking makes the object part of a DLL.

The following graphic illustrates the process for building a business function:



Understanding the Business Function Builder Form

The Business Function Builder form consists of three main components, as described below:

- Functions List
- Available Business Functions
- Build Output

In addition to these components, a status window also appears at the bottom of the form. The status window displays the status of builds as they are in progress. You can click the Stop button in the status bar to discontinue a build.

The combo box in the Toolbar should be set to the Optimize mode. If you are debugging, the box should be set to Debug mode.

Functions List

The Functions List displays the consolidated DLLs for PeopleSoft, including the custom-defined DLLs. It lists the functions that you select to link or compile during a build. Any functions that you drag to the Functions List build when you build your Functions List.

Available Business Functions

Available Business Functions displays the functions that you can build. Each Business Function DLL (for example, CCUSTOM or COBLIB) has its own set of available business functions.

Build Output

Build Output displays build results. As your business functions compile and link, Business Function Builder updates Build Output so that you can see the build in progress. This information is useful if you need to diagnose any problems that occur during a build. When the build is complete, Business

Function Builder indicates whether the build was successful. Use the scroll bars at any time to view the contents of the form. You can also cut and paste text, or print any text that appears in the form.

Building a Single Business Function

You usually use Business Function Builder to build a single business function. Whenever you create source code changes to a business function, you must build the business function to test it.

► To build a single business function

1. On Object Management Workbench, choose the business function that you want to build, and then click the Design button in the center column.
2. On Business Function Design, click the Design Tools tab, and then click Busbuild Standalone.

The BusBuild form appears and Business Function Builder begins building your business function.

Build Output displays the results of the build. When the build is finished, the message *****Build Finished***** appears at the bottom of Build Output. The text following this line indicates whether the build was successful. If the build was successful, you can test your business function. Otherwise, you must correct any problems and rerun the build process.

To stop the build, click the Stop button in the lower right corner of the form. The DLL reverts back to its original form.

Linking Business Function Objects

When you install one or more PeopleSoft EnterpriseOne applications from Object Management Workbench onto your computer, the system also installs business function objects and calls Business Function Builder to perform a Link All operation. This operation integrates the business function objects with your local business function DLLs. Linking preserves your local custom modifications instead of replacing the DLLs. The linking process is unnecessary if you do not have custom modifications on your machine.

Note

Link All does not compile any business functions; it only links each DLL.

Setting Build Options

You can use options on the Build menu to control how and when your consolidated business function is built. The following options are available:

Build	Generates a makefile, compiles the selected business functions, and links the functions into the current consolidated DLL. Rebuilds only those components that are out of date.
Compile	Generates a makefile and compiles the selected business functions. The application does not link the functions into the current consolidated DLL.
ANSI Check	Reviews the selected business function for ANSI compatibility.

Link	Generates a makefile for each consolidated DLL and then builds each consolidated DLL. The application does not compile any of the selected business functions.
Link All	Generates a makefile for each consolidated DLL and then builds each consolidated DLL and links it to all business functions that are called. The application does not compile any of the selected business functions.
Rebuild Libraries	Rebuilds the consolidated DLL and static libraries from the .obj files.
Build All	Links and compiles all objects within each DLL.
Stop Build	Stops the build from finishing. The existing consolidated DLL remains intact.
Suppress Output	Limits the text that appears in Build Output.
Browse Info	Generates browse information when compiling business functions. You can turn this option off to expedite the build.
Precompiled Header	Creates a precompiled header when compiling a business function. When compiling multiple business functions, the Business Function Builder generally compiles faster if it uses a precompiled header.
Debug Info	Generates debug information when compiling. The Visual C++ can debug any function that was built with debug information. You can turn this option off to expedite the build.
Full Bind	Resolves all of the external runtime references for each PeopleSoft EnterpriseOne consolidated DLL.

Noted.Using Utility Programs

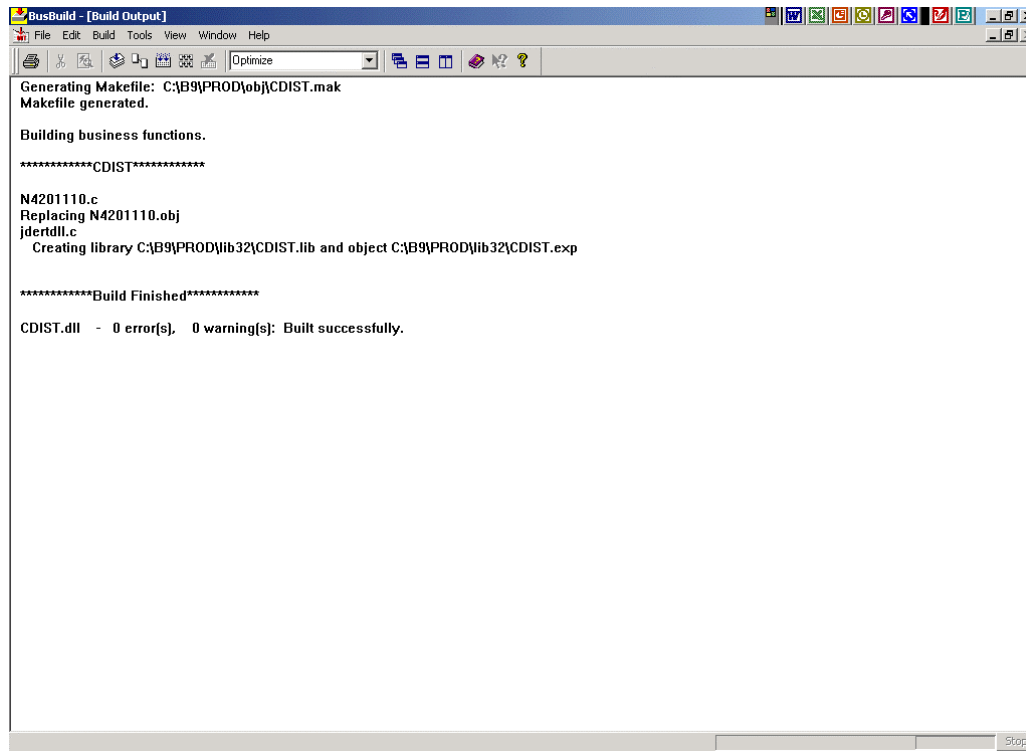
The Tool menu contains several utility programs that assist in the build process. These programs include the following:

Synchronize JDEBLC	You run the Synchronize JDEBLC program to reorganize PeopleSoft EnterpriseOne business functions into new DLL groupings. This program synchronizes DLL field for the local JDEBLC parent specification table with the parent DLL in the Object Librarian Master Table (F9860). Use this program with caution. You typically use this program only if you have manually dragged business function DLLs from a recent package build and you are experiencing failures in the business function load library.
Dumpbin	You run the Dumpbin program to verify whether a particular business function built successfully. This program displays all the business functions that were built into the selected consolidated DLL.
PDB (Program DeBug file) Scan	You receive a CVPACK fatal error when one of the object files that you are trying to link is incorrectly compiled with PDB information. To resolve this problem, you can use the PDB Scan to identify any object fields that were built with PDB information. Recompile any business functions that the PDB Scan reports.

Reading Build Output

Build Output consists of a series of sections that display important information about the status of a build. You can use this information to determine whether your build completed successfully and to troubleshoot problems if errors occurred during your build.

The following graphic is an example of the messages that the system might generate during a build:



Makefile Section

The makefile section indicates where Business Function Builder generated the makefile for a particular build. Business Function Builder generates one makefile for each DLL that it builds. A Generating Makefile statement should always appear for each DLL that you are building. If the makefile statement does not appear, then an error occurred. To resolve the error, you must do the following:

- Verify that the local object directory exists.
- Verify that the permissions for the local object directory and the makefile are correct.

Begin DLL Section

Begin DLL indicates that Business Function Builder is building a particular DLL. For example, assume that the section above this section begins with *****CDIST*****. A Begin DLL section appears for each DLL that you are building.

Compile Section

Before it build DLLs, Business Function Builder compiles the business functions in the DLLs first. The system displays a sequential list of each business function that the Business Function Builder attempts to compile. During the compilation process, the following events might occur:

- **Compiler Warning**

When a compiler warning occurs, Business Function Builder displays `warning cXXXX` (where `XXXX` is a number) and a brief description of the warning. To review information about the warning, search for the `cXXXX` value in Visual C++ online help. Warnings usually do not prevent the business function from compiling successfully. However, you can turn on the Warnings As Errors option in the Global Build form so that the business function will not build if any warnings occur.

- **Compiler Error**

When a compiler error occurs, Business Function Builder displays `error cXXXX` (where `XXXX` is a number) and a brief description of the error. To review extended information about the error, search for the `cXXXX` value in Visual C++ online help. Because errors prevent the business function from compiling successfully, you must resolve them.

Link Section

After Business Function Builder has compiled the business functions for a DLL, it links them. This linking process creates the `.lib` and `.dll` files for the DLL. During linking, the following actions might occur:

- **Linker Warning**

When a linker warning occurs, Business Function Builder displays `warning LNKXXXX` (where `XXXX` is a number) and a brief description of the warning. To review information about the warning, search for the `LNKXXXX` value in the Visual C++ helps. Warnings usually do not prevent the business function from linking successfully. You can turn on the Warnings As Errors option in the Global Build form so that the DLL will not build if it has any warnings occur.

- **Linker Error**

When a linker error occurs, Business Function Builder displays `error LNKXXXX` (where `XXXX` is a number) and a brief description of the error. To review extended information about the error, search for the `LNKXXXX` value in the Visual C++ helps. If a nonfatal error occurs, Business Function Builder still creates the DLL. However, Business Function Builder notes that the DLL was built with errors. If a fatal error occurs, Business Function Builder does not build the DLL.

Rebase Section

The Rebase Section displays information about rebasing. Rebase fine-tunes the performance of DLLs so that they load faster. Rebase does this by changing the desired load address for the DLL so that the system loader does not have to relocate the image. The system automatically reads the entire DLL and also updates fixes, debug information, checksum information, and time stamp values.

Summary Section

The Summary Section contains the most important information about the build. This section indicates whether the build is successful. The summary section begins with "*****Build Finished*****". Business Function Builder also displays a summary report for each DLL that you attempted to build. This report includes the following:

- The number of warnings
- The number of errors
- Whether the DLL build is successful

Building All Business Functions

You can use Build All to build all business functions. Build All performs the same operations as global link, and it recompiles all of the objects within each DLL. A system administrator usually runs Build All. Build All processes can take a long time. To run Build All, you must access BusBuild from the EnterpriseOne Explorer menu.

► To build all business functions

1. From the Windows Start menu, select Run.
2. Type C:\B9\system\Bin32\BusBuild.exe and click OK.
3. On BusBuild, choose Build All from the Build menu.
4. Choose one of the following options for Build Mode:
 - Debug
A build that includes debug information. After you perform a build, you can debug the built business function using the Visual C debugger.
 - Optimize
A build that does not include debug information. Optimized builds generally cannot be debugged using the Visual C debugger.
 - Performance Build
A build that is the same as an optimized build except that it includes information that helps developers measure the performance of business functions. Only PeopleSoft developers should use this option.
5. Complete the Source Directory field.
Use this field to specify where your business function source resides. Business function source includes all .c, .h, named event rules, and table event rules. Full packages usually have all business function sources.
 - a. Click the button to the right of the Source Directory field to browse for a directory.

- b. Choose one of the following locations to store your business function source:

Local	All business function source is on the local machine.
Path Code	All business function source is in the path specified by the selected path code.
Package	The All business function source is in the path specified by the selected package. If a package is built correctly, it typically contains all required business function sources. PeopleSoft recommends that you use Package for your location.
Pick Directory	All business function source is stored in another directory on your file server. You specify the directory.

For example, assume that you have a package called PROD_A that physically resides on the file server location \\SERVER\SHARE1\PROD_A. Assume that PROD_A contains the following subdirectories:

Source	Contains all business function .c.
Include	Contains all business function .h.
Spec	Contains all spec files.

For this configuration on the server for PROD_A, click the button, click Package, and then click PROD_A from the detail area. When Business Function Builder runs, it uses the business function source from this directory.

6. Complete the Foundation Directory field.

Use this field to specify the foundation to use for this build. The foundation that you choose is the foundation on which you expect these business functions to run.

- a. Click the button to the right of this field to browse for a location.
b. Specify your foundation from one of the following locations:

Local	The recommended foundation is the local PeopleSoft EnterpriseOne foundation.
Foundation	The foundation table lists all registered PeopleSoft EnterpriseOne foundations. Choose a foundation from this table.
Pick Directory	The PeopleSoft EnterpriseOne foundation exists in a directory on your file server. You specify the directory. PeopleSoft EnterpriseOne recommends this location.

For example, assume that you have a newly built foundation on the server at location \\SERVER\SHARE1\System\New\Optimize. This directory contains the following subdirectories:

- Lib32** Contains all foundation .lib.
- Include** Contains all foundation .h.
- Includev** Contains all vendor .h.

7. Complete the Output Destination Directory field.

Use this field to specify the location for the output of the build. The build output includes the following file types: DLL, .LIB, .OBJ, .LOG.

- a. Click the button to the right of this field to browse for a location.
- b. Select one of the following locations for your output destination directory:

- Local** All business function output is stored on the local machine.
- Path Code** All business function output is stored in the path specified by the selected pathcode. The pathcode contains the latest business function changes that have been checked in.
- Package** All business function output is stored in the selected package. Package is a more stable snapshot of business function source. PeopleSoft recommends Package for the location of your output destination directory.
- Pick Directory** All business function output is stored in a directory on your file server. You specify the directory.

For example, assume that you have a package called PROD_A that is located on the file server at \\SERVER\SHARE1\PROD_A. When the build finishes, you want Business Function Builder to place the build output in the following subdirectories, which are subordinate to PROD_A:

- Bin32** Contains built .dll files.
- Lib32** Contains built .lib files.
- Obj** Contains built .obj files.
- Work** Contains built .log files.

With the configuration above on the server for PROD_A, click the button, click Package, and then choose PROD_A from the detail area. When Business Function Builder runs, it places the build output files in these subdirectories for PROD_A.

8. Click any of the following options:
- Treat Warnings As Errors

If you enable this option, Business Function Builder does not build a business function if it encounters any warnings.

- Clear Output Destination Before Build

If you enable this option, Business Function Builder deletes the contents of the bin32, lib32 and obj output directories before it builds all business functions.

- Select Which DLLs to Build

If you do not enable this option, Business Function Builder builds all DLLs. If you enable this option, you can click the Select button to choose which business function DLLs you want to build. Use this option if you want to build one or two DLLs. If you build only a subset of all DLLs, verify that the Clear Output Destination Before Build option is disabled.

- Stop Level

You can select the error level at which the build stops. You can ignore errors if you want to continue building despite them. You can specify that the build process stop if a DLL contains errors. You can stop on the first compile error.

- Generate "Missing Source" Report

If you enable this option, Business Function Builder generates a report in the work directory of the destination. This report is called NoSource.txt. It contains business function source file names that do not have a .c file but do have a record in the Object Librarian Master Table (F9860). To resolve the information in this report, you can produce the correct .c file for the business function, or you can delete the source file from the F9860 table. PeopleSoft recommends that you enable this option.

- Generate ER Source

If you enable this option, Business Function Builder generates Named Event Rule and table event rule source before building business functions.

- Verify Check-in

If you enable this option, the system builds only objects checked in to a specified path code. A log file, Notchkdn.txt, is written to the same directory as Nosource.txt. Objects that are not checked in to the path code will be listed in this log and in Buildlog.txt.

You can enable the From RDB option so that you can generate work from any path code. If this option is disabled, the business function builder assumes that the event rules source can be generated from the source directory specification files.

If you are troubleshooting a build initiated by Package Build, then the settings above should already be set to the correct values. In this case, click Build to rebuild the problem DLLs.

Note

You can also run this build by turning the Build BSFN option on in a package build.

Reviewing Error Output

The system creates a work directory when any object is built. This directory is in the destination directory that you specified, such as C:\b7\appl_pgf\work\buildlog.txt.

This directory contains error and information logs. The Buildlog contains the same information as the Build Output form in Business Function Builder.

Resolving Errors

You can use Business Function Builder tools to help you resolve errors. If you notice any unresolved external errors during a business function build, your consolidated DLL still builds, and the software should run normally. However, it cannot execute any unresolved business function.

Use the dumpbin tool to verify that a particular business function is present in a consolidated DLL. If a business function is present, its name appears in the dumpbin output, followed by a nonzero number in parentheses.

Use the PDB scan to resolve the CVPACK fatal error. The CVPACK error occurs when the Business Function Builder attempts to link an object file that was built with PDB (Program DeBug file) information. The PDB scan finds the problem object file. You must then recompile the problem object file on your machine with the Business Function Builder.

If a business function is compiled using Visual C++, it will not work properly. You can use PDB scan to identify any business functions that have been built outside of Business Function Builder. Use Business Function Builder to rebuild these functions so that they work properly.

If one of the DLLs is out of synch, you must rebuild it using the Build option. This generates a makefile and then relinks all the business functions within it.

The Synchronize JDEBLC option from the Business Function Builder Tools menu corrects any misplaced or incorrectly-built business functions. This option reviews the server DLLs and determines whether the local workstation specifications match those of the server. If they do not, then Business Function Builder will rebuild the business functions in the correct DLL on the server and relink them.

The Build Log contains the following sections:

Build Header	This section shows the configuration for a specific build, including the source path, foundation path, and destination path.
Build Messages	This section displays the compile and link activity. During a compile, a line is output for each business function that was compiled. Any compile errors are reported as <i>error cxxxx</i> . During the link part, business function builder outputs the text <i>Creating library . . .</i> . This text might be followed by linker warnings or errors.
Build Summary	The last section of the build summarizes the build for each DLL. This summary is in the form <i>x error(s), x warnings (y)</i> . The summary indicates the status of the build. If you have no warnings and no errors, then the build was successful. If the summary reports an error, search the log for the word <i>error</i> to determine the source of the error. Typical build errors are syntax errors and missing files.

Transaction Master Business Functions

Transaction master business functions provide a common set of functions that contain all of the necessary default values and editing for a transaction table in which records depend on each other. Transaction master business functions contain logic that ensures the integrity of the transaction being inserted, updated, or deleted from the database. Event flow breaks up logic. You use cache APIs to store records that are being processed. You should consider using a transaction master business function in the following situations:

- You accept transaction file records from a non-PeopleSoft EnterpriseOne source.
- Multiple applications update the same transaction file.

The following transaction tables are examples of candidates for transaction master business functions:

- The Account Ledger table (F0911) accepts updates across application suites, as well as external sources.
- The Employee Transaction Detail File table (F06116) accepts updates from batch, interactive, and external sources.

Master Business Function Naming Conventions

The Master Business Function source is named `Bxxxxxxx` (the same as any other business function) unless the object already exists on the iSeries as a functional server, in which case the `XTXXXXZI` naming convention is used. Within this source are external business functions for each of the modules.

The data structure name for each module is `DxxxxxxxA, B, C` and so on. (Examples include B0400047 - D0400047A [Begin Document], D0400047B [Edit Line], D0400047C [Edit Document], D0400047D [End Document].)

The format for naming individual business functions is file-name module-name, and Document is shortened to Doc. For example: F0411BeginDoc.

The functional use code 192 designates business functions as master business functions.

A cache tracks transaction records. It has the following naming convention:

`FxxUIyyy`

Where:

`xx` is the system code

`yyy` is a unique number

Creating Processing Options for a Master Business Function

A Master Business Function (MBF) can be called from several different applications. Rather than duplicating the processing options for the MBF on each application, you typically create a separate processing option template for these processing options.

► To create processing options for a Master Business Function

1. Create a processing option template.

The format for the name of the processing option template is Txxxxxxx (replace the B or XT in the MBF name with a T).

2. Create an artificial application for attaching the MBF processing options.

You can then set up different versions of the MBF processing options through the versions list. The naming convention for this artificial application is Pxxxxxxx (replace the B or XT in the MBF name with a P). To generate, this application needs only one form with no fields selected.

You can use interactive versions to set up different versions of the master business function processing options. Various calling programs then pass the version name to the version parameter of BeginDoc.

From within BeginDoc, the business function AllocatePOVersionData can be called to retrieve the processing options by version name. The processing options needed by other modules can be written to the header cache and accessed later, rather than calling AllocatePOVersionData multiple times.

Naming Transaction Master Business Function Modules

Function names follow the standard naming convention Fxxxxx. If a table has multiple master business functions, include the program name in the function name.

FxxxxxProgram Name Module

For example:

- F3112SuperBackFlushBegDoc
- F3112WorkOrderRoutingsBegDoc

Components of a Transaction Master Business Function

You typically use the following basic components for a master business function:

Begin Document	Called when all header information has been entered • Creates initial header if it has not already been created • Can also include defaulting, editing, and processing options
Edit Line	Called when all line information has been entered • Creates cache for detail information if it has not already been created
Edit Document	Called when ready to commit the transaction • Processes any remaining document edits and verifies that all records are valid to commit
End Document	Called when you need to commit the transaction • Processes all records in the header and detail cache, performs I/O, and deletes caches

Clear Cache	Called when you are ready to delete all cache records
	• Deletes header and detail cache records

Begin Document

Function Name

FxxxxxBeginDocument

What Does It Do?

The Begin Document component does the following:

- Inserts default information and edits information in the header, including data dictionary defaults and UDC editing.
- Fetches information from the database, if necessary, to ensure that the selected document action can take place.
- Validates and processes information that is common to all records.
- Writes the record to header cache if no errors exist.
- Contains all header cache information that is common to all detail records. This improves performance by eliminating the need to use all the detail records to perform the same validations and table I/O.
- Updates the header cache with the new information when information in the header fields changes and Begin Document has previously been called.

Special Logic or Processing Required

On the initial call, the function assigns the job number. To retrieve the job number, this function calls X0010GetNextNumber with a system code of 00 and an index number of 04. If Begin Document is called again, it passes the job number that was previously assigned; therefore, it does not need to assign another job number.

Hook Up Tips

- You must call a function at least once before calling Edit Line.
- If errors occur during validation of the header field when the function is called, call the function again to verify that errors have been cleared before calling Edit Line.
- If this function might be called multiple times from different events, include it on a hidden button on an application to reduce duplicate code and ensure consistency. This button might then be called from focus on grid because the user is then adding or deleting detail records, and is finished adding header information. In case of a Copy in which the user does not use the grid, this button might also be called on OK button.
- Calling a button from an asynchronous event breaks the asynchronous flow and forces the button to be processed in synchronous mode (inline).

Common Parameters

Name	Alias	I/O	Description
Job Number	JOBS	I/O	Pass Job Number created in BeginDocument, if previously called; otherwise, pass zeros and assign a job number.
Document Action	ACTN	I	A or 1 = Add C or 2 = Change D = Delete This is the action of the entire Document, not the individual detail lines. For example, you might modify a few detail lines in Edit Line, add a few detail lines in Edit Line, and delete a few detail lines in Edit Line, but the <i>Document Action</i> in Begin Document would be Change.
Process Edits	EV01	I	<i>Optional</i> 0 = No Edits Any Other = Full Edits Note The GUI interface usually uses the partial edit, and the batch interface uses the full edit. If you leave this parameter blank, the default option is full edits.
ErrorConditions	EV02	O	Blank = No Errors 1 = Warning 2 = Error
Version	VERS	I	This field is required if this MBF is using versions.
Header Field One	****	I/O	Pass in all the header fields that are common to the entire document. Begin Document processes all of these fields and validates them, data dictionary edits, UDC editing, default values, and so on. Begin document might also fetch to the table to validate that records matching these header fields exist for Delete and Change, or do not exist for Add.
Header Field Two	****	I/O	
.	****	I/O	
.			
.			
Header Field XX	****	I/O	

Work Field / Processing Flag One	****	I	List any work fields that the program needs. These could be flags for processing, dates to validate, and so on. These fields might or might not be used. For example, currency control might be saved in the header cache so that all detail records would either use currency or not.
Work Field / Processing Flag One	****	I	
.		I	
Work Field / Processing Flag One		I	

Application-Specific Parameters

- List the fields that are needed to process header level information.
- List any work fields that are needed to perform edits.
- List all processing options that are needed to process header level information.

Edit Line

Function Name

xxxxxxEditLine

What Does It Do?

The Edit Line component does the following:

- Validates all user input, performs calculations, and retrieves default information. Edit Line is normally called for every record that is fetched. It performs the edits for that *one* record in the file.
- Reads header cache records for default values.
- On an ADD, enters default information in blank columns, such as address book information. The default values might come from any of the following:
 - Another column in the line
 - A process performed on a column sent in the line
 - A processing option
 - A saved value from the header record that was determined in the Begin Document module
 - A data dictionary default value
- Edits columns for correct information. This includes interdependent editing between columns. Also performs UDC and data dictionary edits.

- Writes record to the detail cache if no errors occurred. If the record already exists in the work file, the line in the work file will be retrieved and updated with the changes. If a record is deleted from the grid in direct mode, and the record does not exist in the database, the record will be removed from the detail cache. If the record exists in the database, the action code for the record will be changed to delete, and the record will be stored in the detail cache until file processing in End Doc.

Special Logic or Processing Required

The following special processing occurs:

- Depending on the type of document being processed, different editing and inserting of default values takes place. An example would be vouchers and invoices processed through the journal entry MBF. The tax calculator is only called for vouchers.
- Depending on the event processing required, the process edit flag determines the editing that occurs. For example, in an interactive program, when the *Grid Record is Fetched* event runs, Partial Edits might be performed to retrieve descriptions, default values, and so on. When the *Row is Exited and Changed* event runs, Full Edits might be performed to validate all user input.

Typical Uses and Hookup

Interactive	Grid Record is Fetched
	Row is Exited and Changed (Asynch)
Batch	Do section of the group, columnar, or tabular section.

Common Parameters

Name	Alias	I/O	Description
Job Number	JOBS	I	Used as key or to create a unique name for the cache or work file. Retrieved from Begin Document.
Line Number	LNID	I/O	The unique number identifying the transaction line. Can also be used as the line number in the Detail Cache.
Line Action	ACTN	I	A or 1 = Add C or 2 = Change D or 3 = Delete
Process Edits (optional)	EV01	I	0 = No Edits 1 = Full Edits 2 = Partial Edits Note GUI interface typically uses the partial edit, and the batch interface typically uses the full edit. If you leave this parameter blank, the default edit is Full.

Error Conditions	ERRC	O	0 = No Errors 1 = Warning 2 = Error
Update Or Write to Work File	EV02	I	1 = Write or update records to the work file, or do both.
Record Written to Work File	EV03	I/O	1 = A record is written to the work file. This reduces I/O calls to the work file. Blank = No record is written to the work file.
Detail Field One	****	I/O	Pass in all the Detail fields that will be edited. Typically, these are the grid record fields. Edit Line provides validation, data dictionary edits, UDC editing, defaults, and so on.
Detail Field Two	****	I/O	
Detail Field XX	****	I/O	
Work Field / Processing Flag One	****	I	List any work fields that the program needs. These fields could be flags for processing, dates to validate, and so on.
Work Field / Processing Flag One	****	I	
Work Field / Processing Flag One	****	I	

Edit Document

What Does It Do?

The Edit Document component does the following:

- Reads cache records if multiple line editing is required
- Reads header cache record if header information is needed
- Performs cross-dependency edits involving multiple lines in a document. For example, Edit Document processes all records to ensure that percentages total 100 percent, and it ensures that the last record does not contain certain information.

Special Logic or Processing Required

Depending on the type of document that you are processing, different logic is executed. For example, vouchers and invoices are processed through the journal entry edit object, although the balancing is different for these document types.

Hook Up Tips

- Call the function at least once after calling Edit Line and before End Document.

- If errors occur during validation, call the function again to verify that errors have been cleared before calling End Document.
- Call this function on the OK *button clicked* event so that, if errors do occur, they are corrected before the user exits the application.

Common Parameters

Name	Alias	I/O	Description
Job Number	JOBS	I	Retrieved from Begin Document
ErrorConditions	EV01	O	Blank = No Errors 1 = Warning 2 = Error

Application-Specific Parameters

Because all records have been added in Begin Document or Edit Line, and because any information needed to process the entire document is in cache, few parameters are needed in this function.

End Document

Function Name

`FxxxxxEndDocument`

What Does It Do?

The End Document component does the following:

- Assigns a next number to the document. For vouchers, you should do this before calling journal entry edit object, but not before the voucher has been balanced and is ready to be added to the database. By placing this module on the *before add/delete/update* event, the document passes all edits before running this event.
- Reads cache records.
- On an ADD, writes new rows to the table.
- On a CHG, retrieves and updates existing rows.
- On a DEL, deletes rows from the table.
- Adds information and updates associated tables. For example, it adds and updates the following:
 - Manual checks associated with vouchers
 - Address Book vouchered YTD columns in Address Book
 - Address, phones, and who's who information for Address Book
 - Batch header
- Clears the cache for that document and any work fields after all updates are completed successfully.

- Summarizes documents, if designated in a processing option, as it writes to the database. Reads work file through an alternate means and writes the records at a control break.
- Performs currency conversion.

Special Logic or Processing Required

No special logic or processing is required.

Hook-Up Tips

This function is typically called on OK button *Post Button Clicked*, and it is hooked up Asynch. In the C code, after the insert or update to the database is successful, call Clear Cache to clear the cache.

Common Parameters

Name	Alias	I/O	Description
Job Number	JOBS	I	Retrieved from Begin Document
Computer ID	CTID	I	Retrieved from GetAuditInfo(B9800100) in application (optional)
Error Conditions	EV01	O	Blank = No errors 1 = Warning 2 = Error
Program ID	PID	I	Usually hard-coded

Application-Specific Parameters

- List the fields that are needed to process update or writes, such as Time and Date Stamp fields.
- List any work fields that are needed to perform updates or writes.
- List all processing options that are needed to process updates or writes.

Clear Cache

Function Name

FxxxxxClearCache

What Does It Do?

The Clear Cache component removes the records from the header and detail cache

Special Logic or Processing Required

If a unique cache name is selected as the naming convention for the cache (Dxxxxxxx [Job Number]), then use the cache API `jdeCacheTerminateAll` to destroy the cache.

Common Parameters

Name	Alias	I/O	Description
Job Number	JOBS	I	Indicates the job number of the transaction that you want to clear. This job number should have been returned from BeginDoc.
Clear Header	EV01	I	Indicates whether the header cache should be cleared. 1 = clear cache
Clear Detail	EV02	I	Indicates whether the detail cache should be cleared 1 = clear cache
Line Number From (Optional)	LNID	I	Indicates where to begin clearing records in the detail cache. If this line is blank, the system begins clearing from the first record.
Line Number Thru (Optional)	NLIN	I	Indicates where to stop clearing records in the detail cache. If this line is blank, the system deletes to the end of the cache.

Cancel Document

Function Name

xxxxxxCancelDoc

What Does It Do?

The optional Cancel Document component is used primarily with the Cancel button to close files, clear the cache, and so on. Cancel Document is an application-specific function that provides basic function cleanup.

Special Logic or Processing Required

This function is application-specific.

Common Parameters

Name	Alias	I/O	Description
Job Number	JOBS	I	The job number of the transaction that you want to clear. This number should have been returned from BeginDoc.

Cache Structure

The cache structure stores all lines of the transaction. Transaction lines are written to the cache after they have been edited. The EndDoc module then reads the cache to update the database. The cache structure layout is as follows:

Header

Field Description	Field	Key	Type	Size
Job Number	JOBS	X	Num	
Document Action	ACTN		Char	1
Processing Options				
Currency Flag	CYCR		Char	1
Business View Fields				
Work Fields				

Job Number A unique system-assigned number assigned when the BeginDoc module starts the job. This distinguishes transactions in the cache for each job on the workstation that is using the cache. Use next number 00/4 for the job number. If you are using a unique cache name (Dxxxxxxxx [job number]), you do not necessarily need the job number field stored in the cache for a key because you would only be working with one transaction per cache. You can, therefore, use any field as the key to the cache.

Document Action The action for the document. Valid values are:
A or 1 = Add
C or 2 = Change
D = Delete

Processing Options Processing option values were read in using AllocatePOVersionData, and are needed in other modules of the MBF.

Currency Flag A system value that indicates whether currency is on and what method of currency conversion is used (N, Y, or Z).

Business View Fields The fields required for processing the transaction and writing it to the database. All fields in the record format that are not saved in the header cache will be initialized when the record is added to the database using the APIs.

Work Fields Fields that are not part of the business view, but are needed for editing and updating the transaction.

For example, Last Line Number is the last line number written to the detail cache. It will be stored at the header level, and retrieved and incremented by the MBF. The incremented line number will be passed to the header cache and stored for the next transaction.

Detail

Field Description	Field	Key	Type	Size
Job Number	JOBS	X	Char	8
Line Number	(Application-specific)	X	Num	
Line Action	ACTN		Char	1
Business View Fields				
Work Fields				

Job Number A unique number assigned when the BeginDoc module starts the job. This distinguishes transactions in the cache for each job on the client that is using the cache. If you are using a unique cache name (Dxxxxxxx[*job number*]), you do not necessarily need to store the job number field in the cache for a key because you work with only one transaction per cache. You can, therefore, use line number only as the key to the cache.

Line Number The number used to uniquely identify lines in the detail cache. This line number can also eventually be assigned to the transaction when it is written to the database. The transaction lines are written to the detail cache only if they are error-free.

Line Action The action for the transaction line. Valid values are:
A or 1 = Add
C or 2 = Change
D = Delete

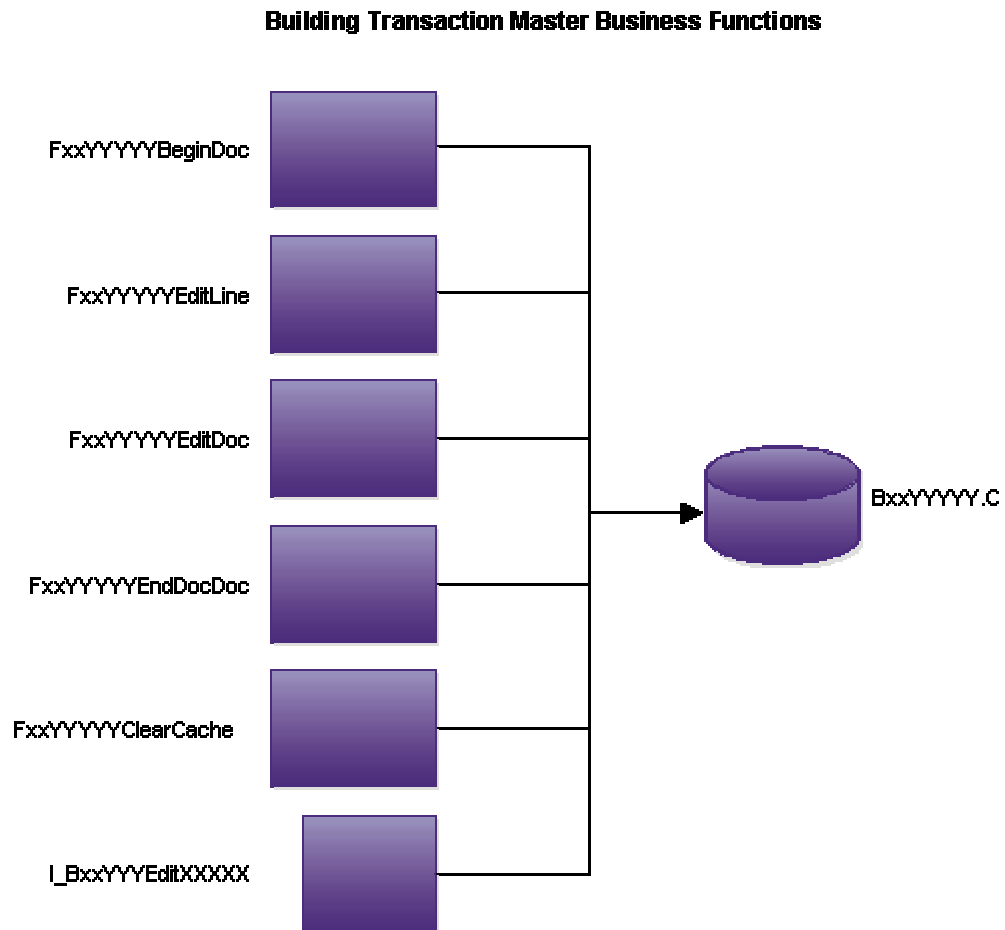
Business View Fields Fields required for processing the transaction that will be written to the database. All fields in the record format that are not saved in the detail cache will be initialized when the record is added to database using the APIs.

Work Fields Fields that are not part of the business view, but are needed for editing and updating the transaction line.

Building Transaction Master Business Functions

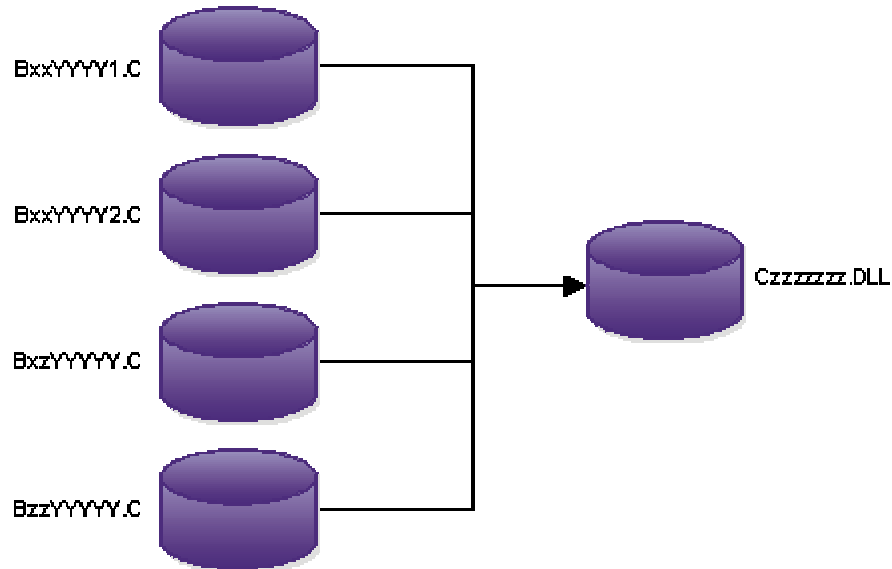
The following graphics show how transaction master business functions are built.

First, you create the individual business functions using the components described previously.



Next, you combine the business functions into a DLL.

Combining Business Functions into a DLL



Implementing Transaction Master Business Functions

You can use single-record processing or document processing to implement transaction master business functions.

Single-Record Processing

Interactive Program Flow Example

- *Post Dialog is Initialized (optional)*
 - Hook up Begin Document function
- *Set Focus on Grid*
- *Row is Exited and Changed or Row is Exited and Changed ASYNC*
 - Hook up Edit Line function
- *Delete Grid Record Verify- After*
 - Hook up Edit Line function to perform delete for one record
 - Hook up Edit Document function to perform deletes on a group of records
- *OK button pressed event*
 - Call Begin Doc
 - Hook up Edit Document function

- *Post OK Button Pressed*
 - Hook up End Document function

Master Business Functions usually perform all table I/O for the given table. Therefore, the following actions must be disabled in the tool:

- *Add Grid Record to DB - before*
 - Suppress Add
- *Update Grid Record to DB - before*
 - Suppress Update
- *Delete Grid Record to DB - before*
 - Suppress Delete

Batch Program Flow Example

- *Do Section of Report Header*
 - Hook up Begin Document function
- *Do Section of the Group Section*
 - Hook up Edit Line function
- *Do Section of a Conditional Section (optional)*
 - Hook up Edit Document function
- *Do Section of Report Footer*
 - Hook up End Document function

Document Processing

Program Flow Example

- *Dialog is Initialized*
 - Hook up Open Batch Edit Object module
- *Grid is entered* - Finished entering in header
 - Hook up Begin Document Edit Object module
- *Row is exited*
 - Hook up Edit Line Edit Object module
- *OK button is pressed*
 - Hook up Edit Document Edit Object module
- *Before Add/Delete from Database event*
 - Suppress Add/Delete
 - Hook up End Document Edit Object module
- *Cancel button is pressed*
 - Hook up Close Batch Edit Object module

Master File Master Business Functions

PeopleSoft provides master business functions (MBFs) to enable calling programs to process certain predefined transactions. An MBF encapsulates the required logic, enforces data integrity, and insulates the calling programs from the database structures.

You can use MBFs for the following reasons:

- To create reusable, application-specific code
- To reduce duplicated code
- To ensure that hookup is consistent
- To support interoperability models
- To allow processing to be distributed through OCM
- To design event-driven architecture

MBFs are typically used for multiline business transactions such as journal entries or purchase orders. However, certain master files also require MBF support due to their complexity, importance, or maintenance requirements from external parties. The requirements for maintaining master files are different from those for multi-line business transactions.

Generally, master file MBFs are much simpler than multi-line business transaction MBFs. Transaction MBFs are specific to a program, while master file MBFs access a table multiple times.

For interoperability, master file MBFs can be used instead of table I/O. This allows you to perform updates to related tables using the business function instead of table event rules. Multiple records are not used; instead, all edits and actions are performed with one call.

In their basic form, master file MBFs have the following characteristics:

Single call	Generally, you can make one call to an MBF to edit, add, update, or delete a master file record. An edit-only option is also available.
Single data structure	The fields required to make the request and provide all the necessary values are in one data structure. The data fields should correspond directly with columns in the associated master file.
No cache	Because each master file record is independent of the others, caching is unnecessary. The information provided with each call and the current condition of the database provides all of the information that the MBF needs to perform the requested function.
Normal error handling	As with other MBFs, master file MBFs must be capable of executing both in interactive and batch environments. Therefore, the calling program must determine the delivery mechanism of the errors.
Inquiry feature	To allow external systems to be insulated from the PeopleSoft EnterpriseOne database, an inquiry option is included. This allows an external system to use the same interface to access descriptive information about a master file key as it uses to maintain it.
Effect on applications	For PeopleSoft EnterpriseOne applications, the effect of implementing a master file MBF should be minimal. Consider and follow several standards before implementing a master file MBF.

Master file applications use the system to process all I/O for the Find/Browse forms. This allows you to use all of the search capabilities of the software.

You should design all master file applications so that all fix/inspect forms are independent of each other. Each fix/inspect form can use the system to fetch the record, and all edits and updates occur using the master file MBF. This independent design has the following major benefits:

- It organizes the application in a way that simplifies edits involving dependent fields across multiple forms.
- It allows consistent implementation of modeless processing for all master file applications and all forms within these applications.

Certain circumstances might justify deviation from this simple model. These circumstances are as follows:

- Extremely large file formats

When the number of columns in the master file plus the required control fields in the call data structure exceed technical limitations for data structures, the MBF can be split. PeopleSoft recommends that you split the MBF into one MBF that handles base data and performs all adds and deletes, and one or more MBFs that allow the calling program to update additional data when the base data has been established. In this case, it is usually logical to split it, regardless of the technical limitation. For example, assuming that the customer master file exceeded the data structure limitation, you would use the following two MBFs to process the file:

- F0301ProcessMasterData
- F0301ProcessBillingData

In this example, the F0301ProcessMasterData function processes the base data, and the F0301ProcessBillingData function updates additional data.

- Subordinate detail files

Information can exist in addition to the primary master file that has been normalized to allow for a one-to-many relationship. Designing the Master File MBF strictly on the basis of how the database is designed translates into three calls. Including at least one occurrence of a detail relationship in the data structure of a Master File MBF is valid. This inclusion allows users to establish reasonably complete master file information using a simple interface to meet simple needs. Street addresses and phone numbers within Address Book are a good example. Customers expect that they can create an address book record by calling a simple address book API with basic identifying information, the street address, and a phone number.

Information Structure

Standard Parameters for Single-Record Master Business Functions

Name	Alias	I/O	Required/Optional	Description
Action Code	ACTN	I	Required	A = Add I = Inquiry C = Change D = Delete S = Same as except (The record is the same except for what the user changes.)
Update Master File	EV01	I	Optional	0 = No update edit only (Default) 1 = Update performed
Process Edits	EV02	I	Optional	1 = All Edits (Default) 2 = Partial Edits (No Data Dictionary)
Suppress Error Messages	SUPPS	I	Optional	1 = Error Messages are Suppressed 0 = Process Errors Normally (Default)
Error Message ID	DTAI	O	Optional	Returns error code
Version	VERS	I	Future	Default to XJDE0001

Application-Specific Control Parameters (Example: Address Book)

Name	Alias	I/O	Required/Optional	Description
Address Book Number	AN8	I/O	Optional	For additions, AN8 is optional. For all other action codes, this parameter is required.
Same as except	AN8	I	Optional	Required for S = Action Code The record is the same except for what the user changes.

Application Parameters (Example: Address Book)

Name	Alias	I/O	Required/Optional
Alpha Name	ALPH	I/O	Required
Long Address Number	ALKY	I/O	Optional
Search Type	AT1	I	Required
Mailing Name	MLMN	I	Required

Address Line 1	ADD1	I	Optional
City	CTY1	I	Optional
State	ADD5	I	Optional
Postal Code	ADDZ	I	Optional

Naming Convention

- The functional server source is named `Nxxxxxxx` for named event rule business functions or `Bxxxxxxx` for C business functions, which is the same naming convention as that for any other business function. Within this source is an external business function for each of the modules. The data structure name for each module is `DxxxxxxA`, `B`, `C` and so on (for example, `N03000052 - D03000052A F0301ProcessMasterData`, `D03000052B F0301ProcessBillingData`).
- The individual business function is named *file name*`ProcessMasterData` (for example, `F0301ProcessMasterData`). If additional master file functions are needed due to data structure limitations, then the function name is named *file name*`Processxxxxxxx`, where `xxxxxxx` represents a descriptive name for the type of data being processed (for example, `F0301ProcessBillingData`).
- The functional use code of 192 in the Object Librarian is used to designate business functions as a functional server.

Impact on Performance

Performance issues might occur regardless of how you handle large-format tables. Two options for improving performance are:

- Group data logically to allow data structures to be smaller and easier for the user to implement. This configuration does, however, force the user to make multiple calls to add or update an entire record in a table.
- Use a data structure that allows 300 fields. This configuration is cumbersome to implement, and the user can choose not to apply all of the fields.

Through different interfaces, the user can add additional data later. Most processes dictate that part of the data be added immediately, while related data can be added later. For example, the user might define a customer master record but wait until a later date to define the customer's billing instructions. Therefore, PeopleSoft recommends that you choose the first option of splitting MBFs so that one MBF handles base data and one MBF handles additional data.

Business Function Documentation

Business function documentation explains what individual business functions do and how they should be used. The documentation for a business function should include information such as the following:

- Purpose
- Parameters (the data structure used)

- Explanation of each individual parameter that indicates required input and output, and an explanation of return values
- Related tables (the table accessed)
- Related business functions (business functions called from within the functions itself)
- Special handling instructions

You use Business Function Design and Data Structure Design to document your business functions.

Creating Business Function Documentation

You can create business function documentation for several levels, including the following:

Business Function Notes	Documentation for the specific business function that you are using
Data Structure Notes	Notes about the data structure for the business function
Parameter Notes	Notes about the actual parameters in the data structure

► To create business function documentation

1. On Object Management Workbench, choose the business function that you want to document, and then click the Design button in the center column.
2. On Business Functions Design, click the Attachments tab.
3. Enter your documentation in the Attachments pane.
Alternatively, you can use a template ensure consistency. To use a template, complete steps 4 through 6.
4. Right-click in the icon bar on the left side of the form, and then choose Templates.
5. On Work With Media Object Templates, click Find.
Alternatively, click the Add button to add your own template at this point. You must add your template and then complete steps 5 through 7.
6. Select the template that you want to use.
PeopleSoft EnterpriseOne uses the Business Function template as a standard.
7. Type in the appropriate information under each template heading.

Business Function Documentation Template

The Business Function documentation template contains the following sections:

Purpose	This section contains a brief summary of what the function does.
Setup Notes and Prerequisites	This section includes any special notes to assist in using the function, including prerequisite functions, special values that need to be initialized, and events that are recommended to run the function. It also indicates whether memory must be cleared separately after the function is used.
Special Logic	This section contains additional details about the business function logic. You typically use it only for complex functions that require more explanation than the purpose summary.
Technical Specification	This section contains the technical specification of the function, written in scripted English. It can be a direct copy from an existing word processing document.

Creating Data Structure Documentation

Use the following process to create data structure documentation.

► To create data structure documentation

1. On Object Management Workbench, check out the data structure that you want to document.
2. Ensure that the data structure is highlighted, and then click the Design button in the center column.
3. On Data Structure Design, click the Attachments tab.
4. Enter your documentation in the Attachments pane.

Alternatively, right-click in the icon bar on the left side of the form, and then select Templates. Select and complete a template.

Data Structure Documentation Template

The data structure documentation template contains the following sections:

Special Input Expected	Delete this section if it does not apply.
Special Output Returned	Delete this section if it does not apply.
Significant Data Values	Delete this section if it does not apply.

Creating Parameter Documentation

The steps for creating parameter documentation are the same as those for creating data structure documentation, with one exception. After selecting the data item in the structure for which you want to enter notes, click the Data Structure Item Attachments binder clip. Parameter documentation has no special template.

You can enter specific notes about a data structure item to further clarify the information that should be passed in or passed out of the item, such as a mode parameter. The notes should indicate the valid values that the function accepts when you hook it up and information about how to use these values. For example, valid values might include 1 for Add and 2 for Delete. You include that information in parameter documentation.

Generating Business Function Documentation

Generating business function documentation provides you with an online list of information about business functions that you can view through the Business Function Documentation Viewer (P98ABSFN). Typically, the system administrator performs this task because generating the business function documentation for all business functions takes considerable time. If you create new business function documentation, you need to regenerate the business function documentation for that business function only.

► To generate business function documentation

From the Cross Application Development Tools menu (GH902), select Generate BSFN Documentation.

1. On Work with Batch Versions, choose version XJDE0001.
2. On Version Prompting, if you do not want to generate all business function documentation, choose the following option:
 - Data Selection
3. Click Submit.
4. On Data Selection, build your criteria for data selection, and then click OK.

Choose only those functions for which you are generating documentation.

Note

Depending on the criteria that you choose, you might also need to set processing options.

5. On Report Output Destination, if you are running your report locally, choose one of the following output destinations:
 - On Screen
 - To Printer

The system creates a hypertext markup language (HTML) link for each business function for which you generated documentation. It also creates an Index HTML file. These HTML files appear in your output queue directory. Output is in the following format:

```
Function Name
Function Description from
Parent DLL:
Location:
Language:
Purpose
Special Handling
Data Structure
Parameter Name dataitem data type req/ opt
```

Data Selection Tips

You can use data selection to choose the business functions for which you wish to generate documentation. Generate BSFN Documentation (R98ABSFN) uses your data selection criteria to filter the business function documentation. The report might take considerable time to run when you generate documentation for all business functions. You can use data selection to generate documentation for one business function, all business functions, or any combination of business functions.

For example, if you want to generate documentation for a single business function, you can use the data item BC Object Name (F9860).

Operator	Left operand	Comparison	Right operand
Where	BC Object Name (F9860) (OBNM)	is equal to	"B76C0021"

If you want to generate documentation for all of the business functions for a specific product code, such as Payroll, you use the data item BC Product Code (F9860).

	Operator	Left operand	Comparison	Right operand
	Where	BC Product Code (F9860) (SY)	is equal to	"07"

You can also use the right operand on the Data Selection form to choose ranges or lists of values to further refine your data selection.

You can use any value that is associated with a business function to narrow the range of business functions for which you want to generate documentation. For example, you can use BC Date - Updated (F9860) if you have already produced the documentation for a previous software release and, after a software upgrade or update, you want to create documentation for only new or modified business functions. Other examples include the following:

- Use BC Function Type (F9860) to choose Master business function documentation.
- Use BC Location Business Function (F9860) to produce documentation for client-run business functions.
- Use BC Object Type (F9860) to generate documentation for NERs only.

Viewing Documentation from Business Function Search

When you connect to a business function in event rules, the Business Function Search form appears. You can then select the function that you want to call. From the row menu, choose Data Structure Notes or Attachments to view the documentation for the business function.

Viewing Documentation from Business Function - Values to Pass

You can click any of the following buttons on Business Function - Values to Pass to view documentation for a single business function.

BSFN Notes Displays the notes for the business function

Structure Notes Displays the notes for the whole data structure

Parameter Notes Displays the notes for a particular parameter

See Also

- ❑ *Business Function Event Rules* in the *Development Tools Guide* for more information about accessing the Business Function - Values to Pass form

Viewing Documentation from Business Function Documentation Viewer

You can use Business Function Documentation Viewer to view documentation for all business functions or selected business functions. After you generate your report, use the Business Function Documentation Viewer (P98ABSFN) to display your information. PeopleSoft suggests that you use this method to view business function documentation.

The Business Function Documentation form contains the HTML index that you generated. You can either view the entire index or choose specific functions by clicking the appropriate letter in the index. Double-click a business function to view documentation that is specific to that function.

The media object loads the HTML index of the business functions based on a media object queue. In the media object queue table, a queue named Business Function Doc is defined.

This queue must point to the directory in which the business function HTML files are located. The system administrator usually generates the documentation for all business functions. Because the generation process places the documentation files in the local directory, the administrator must then copy the files to a central directory on the deployment server. The files must be copied to the media object queue for media object business function notes. If you are using the standalone version of the software, this path is usually the output directory from the Network Queue Settings section of your jde.ini file. If this entry is not in your jde.ini file, it is in the print queue directory in your PeopleSoft EnterpriseOne software directory.

Note

See the *System Administration Guide* for more information about defining a media object queue.

► To view documentation from Business Function Documentation Viewer

From the Cross Application Development Tools menu (GH902), choose Business Function Documentation Viewer (R9000C).

View the business function documentation HTML index that you generated.

Understanding Business Function Processing Failovers

In some instances in which a business function fails to process correctly, the software can attempt to recover and reprocess the transaction. The system recognizes two principle failure states: process failure and system failure.

A process failure occurs when a jdenet_k process aborts abnormally. For a process failure, the software server processing launches a new jdenet_k process and continues processing.

A system failure occurs when all the server processing fails, the machine itself is down, or the client cannot reach the server because of network problems. For a system failure, business function processing must be rerouted either to a secondary server or to the local client. The system uses the following process to attempt to recover from this state:

When the call to the server fails, the system attempts to reconnect to the server.

- If reconnect succeeds and no cache exists, the system reruns the business function on the server. If a cache does exist, the system forces the user out of the application.
- If reconnect fails and no cache exists, the system switches to a secondary server or to the local client. If a cache does exist, the system forces the user out of the application.

After one module switches, all subsequent modules switch to the new location.

Caching

Caching is a process that stores a local copy of frequently accessed content of remote objects. Caching can improve performance. EnterpriseOne software caches information in the following two ways:

- The system automatically caches some tables, such as those associated with constants, when it reads them from the database at startup. It caches these tables to a user's workstation or to a server for faster data access and retrieval.
- Individual applications can be enabled to use cache. JDECACHE APIs allow the server or workstation memory to be used as temporary storage.

JDECACHE

JDECACHE is a component of JDEKRNL that can hold any type of indexed data that your application needs to store in memory, regardless of the platform on which the application is running; therefore, an entire table can be read from a database and stored in memory. No limitations exist regarding the type of data, size of data, or number of data caches that an application can have, other than the limitations of the computer on which it is running. Both fixed-length and variable-length records are supported. To use JDECACHE on any supported platform, you need to know only a simple set of API calls.

Data handled by JDECACHE is in RAM. Therefore, ensure that you really need to use JDECACHE. If you use JDECACHE, design your records and indices carefully. Minimize the number of records that you store in JDECACHE because PeopleSoft EnterpriseOne software and various other applications need this memory as well.

JDECACHE supports multiple cursors, multiple indexes, and partial keys processing. JDECACHE is flexible in terms of positioning within the cache for data manipulation, which improves performance by reducing searching within the cache.

When to Use JDECACHE

The following scenario describes when an application might use the JDECACHE APIs.

You use workfiles when an application must store records that a user enters in a detail area until OK processing is activated upon the *Button Clicked* event. On OK processing, all records must be simultaneously updated to the database. This is similar to transaction processing. For example, in the detail area of purchase order detail, if a user enters 30 lines of information and then decides to cancel the transaction, all records in the workfile are deleted and nothing is written to the database. As the user exits each detail row, editing takes place for each field, and then that record is written to the workfile.

If you implement this situation without using workfiles, irreversible updates to database tables occur when the user exits each row. Using workfiles allows you to limit updates to tables so that they only occur on OK button processing, and they are included in a transaction boundary. The workfile defines a data boundary for the grid for processing purposes. This is useful when multiple applications or

processes (such as business functions) must access the data in the workfile for updates and calculations.

Although using workfiles for the example above will function correctly, using cache will probably increase performance. You can use JDECACHE to store in memory the records that the user enters in one purchase order. The number of records that you store depends on the cache buffer size for each record, the local memory size, the location in which the business function that you use runs (for example, server or workstation), and so on. Typically, you should not store more than 1000 records. For example, do not cache your entire Address Book table in memory.

Performance Considerations

The following guidelines give the best JDECACHE performance:

- Cache as few records as possible.
- The fewer columns (segments) that you use, the faster your search, insert, and delete actions occur. In some cases, the system might have to compare each column before it determines whether to go further in the cache.
- The fewer records in your cache, the faster all operations proceed.

The JDECACHE API Set

You use a set of public APIs to interact with JDECACHE. You must understand how the JDECACHE APIs are organized to implement them effectively.

JDECACHE Management APIs

You can manage cache using the JDECACHE management APIs for the following:

- Setting up the cache
- Clearing the cache
- Terminating the cache

Use the `jdeCacheGetNumRecords` and `jdeCacheGetNumCursors` APIs to retrieve cache statistics. They are only passed the `HCACHE` handle. All other JDECACHE management APIs should always be passed the following handles:

- `HUSER`
- `HCACHE`

These two handles are essential for cache identification and cache management.

The set of JDECACHE management APIs consist of the following APIs:

- `jdeCacheInit`
- `jdeCacheInitMultipleIndex`
- `jdeCacheInitUser`
- `jdeCacheInitMultipleIndexUser`
- `jdeCacheGetNumRecords`

- jdeCacheGetNumCursors
- jdeCacheClear
- jdeCacheTerminate
- jdeCacheTerminateAll

The jdeCacheInit/jdeCacheInitMultipleIndex API initializes the cache uniquely per user. Therefore, if a user logs in to the software and then runs two sessions of the same application simultaneously, the two application sessions will share the same cache. Consequently, if the first application deletes a record from the cache, the second application cannot access the record. Conversely, if two users log in to the software and then run the same application simultaneously, the two application sessions have different caches. Consequently, if the first application deletes a record from its cache, the second application will still be able to access the record in its own cache.

The jdeCacheInitUser/jdeCacheInitMultipleIndexUser API initializes the cache uniquely per application. Therefore, if a user logs in to the software and then runs two sessions of the same application simultaneously, the two application sessions will have different caches. Consequently, if the first application deletes a record from its cache, the second application can still access the record in its own cache.

JDECACHE Manipulation APIs

You can use the JDECACHE manipulation APIs for retrieving and manipulating the data in the cache. Each API implements a cursor that acts as pointer to a record that is currently being manipulated. This cursor is essential for navigation within the cache. JDECACHE manipulation APIs should be passed handles of the following types:

HCACHE Identifies the cache that is being worked

HJDECURSOR Identifies the position in the cache that is being worked

The set of JDECACHE manipulation APIs is as follows:

- jdeCacheOpenCursor
- jdeCacheResetCursor
- jdeCacheAdd
- jdeCacheFetch
- jdeCacheFetchPosition
- jdeCacheUpdate
- jdeCacheDelete
- jdeCacheDeleteAll
- jdeCacheCloseCursor
- jdeCacheFetchPositionByRef
- jdeCacheSetIndex
- jdeCacheGetIndex

Working with JDECACHE

The JDB environment creates, manages, and destroys the JDECACHE environment. Each cache that you use within the JDECACHE environment is associated with a JDB user. Therefore, you must call JDB_InitBhvr API before you call any of the JDECACHE APIs.

Before you can use JDECACHE, you must initialize a cache. You must define an index before you initialize a cache. The index specifies to the cache which fields in a record are used to uniquely identify a cache record. You must create a separate cache for each group of data that an index references.

Calling JDECACHE APIs

JDECACHE APIs must be called in a certain order. The following bullets list the order in which the JDECACHE-related APIs must be called. At JDECACHE Operations, the actual JDECACHE APIs can be called in any order:

1. JDB_InitBhvr
2. Create index or indices
3. jdeCacheInit or jdeCacheInitMultipleIndex
4. jdeCacheAdd
5. jdeCacheOpenCursor
6. JDECACHE Operations

The following indented list of JDECACHE operations can occur in any order:

- jdeCacheFetch
- jdeCacheOpenCursor (the second cursor)
- jdeCacheFetchPosition
- jdeCacheUpdate
- jdeCacheDelete
- jdeCacheDeleteAll
- jdeCacheResetCursor
- jdeCacheCloseCursor (if the second cursor is opened)
- jdeCacheCloseCursor
- jdeCacheTerminate
- JDB_FreeBhvr

Setting Up Indices

To store or retrieve any data in JDECACHE, you must set up at least one index that consists of at least one column. The index is limited to a maximum of 25 columns (which are called segments) in the index structure. Use the data type provided to tell the cache manager what your index looks like. You must provide the number of columns (segments) in the index and the offset and size of each column in your data structure. To maximize performance, minimize the number of segments.

The following is the definition of the structure that holds index information:

```
#define JDECM_MAX_NUM_SEGMENTS 25

struct _JDECMKeySegment
{
    short int nOffset;          /* Offset from beginning of structure in bytes */
    short int nSize;           /* Size of data item in bytes */
    int idDataType;            /* EVDT_MATH_NUMERIC or EVDT_STRING*/
} JDECMKEYSEGMENT;

struct _JDECMKeyStruct
{
    short int nNumSegments;

    JDECMKEYSEGMENT CacheKey[JDECM_MAX_NUM_SEGMENTS];
} JDECMINDEXSTRUCT;
```

Observe the following rules when you create indices in JDECACHE:

- Always declare the index structure as an array that holds one element for single indexes. Declare the index structure as an array that holds more than one element for multiple indexes. You can create an unlimited number of indexes.
- Always use `memset()` for the index structure. When you use `memset()` for multiple indexes, multiply the size of the index structure by the total number of indexes.
- Always assign as elements the number of segments that correspond to the number of columns that you have in the `CacheKey` array.
- Always use `offsetof()` to indicate the offset of a column in the structure that contains the columns.

The following example illustrates a single index with multiple fields:

```
/* Example of single index with multiple fields.*/

JDECMINDEXSTRUCT      Index[1]                = {0};

memset (&dsCache, 0x00, sizeof (dsCache) );

/* Initialize cache. */
```

```

Index->nNumSegments=5;
Index->CacheKey[0].nOffset=offsetof(DSCACHE,szEdiUserId);

Index->CacheKey[0].nSize=DIM(dsCache.szEdiUserId);
Index->CacheKey[0].idDataType=EVDT_STRING;


Index->CacheKey[1].nOffset=offsetof(DSCACHE,szEdiBatchNumber);
Index->CacheKey[1].nSize=DIM(dsCache.szEdiBatchNumber);
Index->CacheKey[1].idDataType=EVDT_STRING;


Index->CacheKey[2].nOffset=offsetof(DSCACHE,szEdiTransactNumber);
Index->CacheKey[2].nSize=DIM(dsCache.szEdiTransactNumber);
Index->CacheKey[2].idDataType=EVDT_STRING;


Index->CacheKey[3].nOffset=offsetof(DSCACHE,mnEdiLineNumber);
Index->CacheKey[3].nSize=sizeof(dsCache.mnEdiLineNumber);
Index->CacheKey[3].idDataType=EVDT_MATH_NUMERIC;


Index->CacheKey[4].nOffset=offsetof(DSCACHE,cErrorCode);
Index->CacheKey[4].nSize = 1;
Index->CacheKey[4].idDataType=EVDT_CHAR

```

The flag `idDataType` indicates the data type of the particular key.

The following example illustrates a cache with multiple indices and multiple fields:

```

Memset(jdecIndex,0x00,sizeof(JDECMINDEXSTRUCT)*2);

jdecIndex[0].nKeyID=1;
jdecIndex[0].nNumSegments=6;
jdecIndex[0].CacheKey[0].nOffset=offsetof(I1000042,szCostCenter);
jdecIndex[0].CacheKey[0].nSize=DIM(dsI1000042.szCostCenter);
jdecIndex[0].CacheKey[0].idDataType=EVDT_STRING;
jdecIndex[0].CacheKey[1].nOffset=offsetof(I1000042,szObjectAccount);
jdecIndex[0].CacheKey[1].nSize=DIM(dsI1000042.szObjectAccount);
jdecIndex[0].CacheKey[1].idDataType=EVDT_STRING;
jdecIndex[0].CacheKey[2].nOffset=offsetof(I1000042,szSubsidiary);
jdecIndex[0].CacheKey[2].nSize=DIM(dsI1000042.szSubsidiary);
jdecIndex[0].CacheKey[2].idDataType=EVDT_STRING;

```

```

jdecIndex[0].CacheKey[3].nOffset=offsetof(I1000042,szSubledger);
jdecIndex[0].CacheKey[3].nSize=DIM(dsI1000042.szSubledger);

jdecIndex[0].CacheKey[3].idDataType=EVDT_STRING;

jdecIndex[0].CacheKey[4].nOffset=offsetof(I1000042,szSubledgerType);
jdecIndex[0].CacheKey[4].nSize=1;

jdecIndex[0].CacheKey[4].idDataType=EVDT_STRING;

jdecIndex[0].CacheKey[5].nOffset=offsetof(I1000042,szCurrencyCodeFrom);
jdecIndex[0].CacheKey[5].nSize=DIM(dsI1000042.szCurrencyCodeFrom);
jdecIndex[0].CacheKey[5].idDataType=EVDT_STRING;

***** KEY 2 *****

jdecIndex[1].nKeyID=2;
jdecIndex[1].nNumSegments=7;

jdecIndex[1].CacheKey[0].nOffset=offsetof(I1000042,szEliminationGroup);
jdecIndex[1].CacheKey[0].nSize=DIM(dsI1000042.szEliminationGroup);
jdecIndex[1].CacheKey[0].idDataType=EVDT_STRING;

jdecIndex[1].CacheKey[1].nOffset=offsetof(I1000042,szCostCenter);
jdecIndex[1].CacheKey[1].nSize=DIM(dsI1000042.szCostCenter);
jdecIndex[1].CacheKey[1].idDataType=EVDT_STRING;

jdecIndex[1].CacheKey[2].nOffset=offsetof(I1000042,szObjectAccount);
jdecIndex[1].CacheKey[2].nSize=DIM(dsI1000042.szObjectAccount);
jdecIndex[1].CacheKey[2].idDataType=EVDT_STRING;

jdecIndex[1].CacheKey[3].nOffset=offsetof(I1000042,szSubsidiary);
jdecIndex[1].CacheKey[3].nSize=DIM(dsI1000042.szSubsidiary);
jdecIndex[1].CacheKey[3].idDataType=EVDT_STRING;

jdecIndex[1].CacheKey[4].nOffset=offsetof(I1000042,szSubledger);
jdecIndex[1].CacheKey[4].nSize=DIM(dsI1000042.szSubledger);
jdecIndex[1].CacheKey[4].idDataType=EVDT_STRING;

jdecIndex[1].CacheKey[5].nOffset=offsetof(I1000042,szSubledgerType);
jdecIndex[1].CacheKey[5].nSize=1;
jdecIndex[1].CacheKey[5].idDataType=EVDT_STRING;

jdecIndex[1].CacheKey[6].nOffset=offsetof(I1000042,szCurrencyCodeFrom);
jdecIndex[1].CacheKey[6].nSize=DIM(dsI1000042.szCurrencyCodeFrom);
jdecIndex[1].CacheKey[6].idDataType=EVDT_STRING;

```


Initializing the Cache

After you set up the index or indices, you call `jdeCacheInit` or `jdeCacheInitMultipleIndex` to initialize (create) the cache.

Pass a unique cache name so that JDECACHE can identify the cache. Pass the index to this API so that the JDECACHE knows how to reference the data that will be stored in the cache. Because each cache must be associated with a user, you must also pass the user handle obtained from the call to `JDB_InitUser`. This API returns an HCACHE handle to the cache that JDECACHE creates. This handle appears in every subsequent JDECACHE API to identify the cache.

The keys in the index must always be the same for every `jdeCacheInit` and `jdeCacheInitMultipleIndex` call for that cache until it is terminated. The keys in the index must correspond in number, order, and type for that index each time that it is used.

After the cache has been initialized successfully, JDECACHE operations can take place using the JDECACHE APIs. The cache handle obtained from `jdeCacheInit` must be passed for every JDECACHE operation.

JDECACHE makes an internal Index Definition Structure that accesses the cache when it is populated. The following topic provides an example that illustrates what happens when an index is defined and passed to `jdeCacheInit`.

Example: Index Definition Structure

In this scenario, you decide that each record that the cache stores has the following structure:

```
int nInt1  
  
JCHAR cLetter1  
  
JCHAR cLetter2  
  
JCHAR cLetter3  
  
JCHAR szArray(5)
```

You decide that each record in the cache will be indexed uniquely by the values contained in the following:

- `nInt1`
- `cLetter1`
- `cLetter3`

You pass that information to `jdeCacheInit`, and JDECACHE creates the following Index Definition Structure for internal use. The Index Definition Structure is for STRUCT letters.

Example: Index Definition Structure

Index Key No. Index Key #1	Index Key Offset 0	Index Key Offset INTEGER
Index Key #2	4	JCHAR
Index Key #3	6	JCHAR

Using an Index to Access the Cache

When you use an index to access the cache, the keys in the index that are sent to the API must correspond to the keys of the index used in the call to `jdeCacheInit` for that cache in number, order, offset positions, and type. Therefore, if a field that was used in the index passed to `jdeCacheInit` offsets position 99, it must also offset position 99 in the index structure that passed to JDECACHE access API.

You should use the same index structure that was used for the call to `jdeCacheInit` whenever you call an API that requires an index structure.

The example in the following topic illustrates why the index offsets must be specified for the `jdeCacheInit` and how they are used when a record is to be retrieved from the cache. It shows how the passed key is used in conjunction with the JDECACHE internal index definition structure to access cache records.

Example: JDECACHE Internal Index Definition Structure

In this example, assume that the user is looking for a record that matches the following index key values:

- 1
- c
- i

JDECACHE accesses the values that you pass in the structure at the byte offsets that were defined in the call to `jdeCacheInit`.

JDECACHE compares the values 1, c, and i that it retrieves from the passed structure to the corresponding values in each of the cache records at the corresponding byte offset. The cache records

are stored as the structures that were inserted into the cache by `jdeCacheAdd`, which is the same structure as the one you pass first. The structure that matches the passed key is the second structure to which `HCUR1` points.

You should never create a smaller structure that contains just the key in order to access the cache. Unlike most indexing systems, JDECACHE does not store a cache record's index separately from the actual cache record. This is because JDECACHE deals with memory-resident data and is designed to be as memory-conservative as possible. Therefore, JDECACHE does not waste memory by storing an extra structure for the sole purpose of indexing. Instead, a JDECACHE record has a dual purpose of index storage and data storage. This means that, when you retrieve a record from JDECACHE using a key, the key should be contained in a structure that is of the same type as the structure that is used to store the record in the cache.

Do not use any key structure to access the cache other than the one for which offsets that were defined in the index passed to `jdeCacheInit`. The structure that contains the keys when accessing a cache should be the same structure that is used to store the cache records.

If `jdeCacheInit` is called twice with the same cache name and the same user handle without an intermediate call to `jdeCacheTerminate`, the cache that was initialized using the first `jdeCacheInit` will be retained. Always call `jdeCacheInit` with the same index each time that you call it with the same cache name. If you call `jdeCacheInit` for the same cache with a different index, none of the JDECACHE APIs will work.

The key for searches must always use the same structure type that stores cache records.

Using the `jdeCacheInit/jdeCacheTerminate` Rule

For every `jdeCacheInit` or `jdeCacheInitMultipleIndex`, a corresponding `jdeCacheTerminate` must exist, except instances in which the same cache is used across business functions or forms. In this case, all unterminated `jdeCacheInit` or `jdeCacheInitMultipleIndex` calls must be terminated with a `jdeCacheTerminateAll`.

A `jdeCacheTerminate` call terminates the most recent corresponding `jdeCacheInit`. This means that the same cache can be used in nested business functions. In each function, perform a `jdeCacheInit` that passes the cache name. Before exiting that function, call `jdeCacheTerminate`. This does not destroy the cache. Instead, it destroys the association between the cache and the passed `HCACHE` handle. The cache is completely destroyed from memory only when the number of `jdeCacheTerminate` calls matches the number of `jdeCacheInit` calls. In contrast, one call to `jdeCacheTerminateAll` destroys the cache from memory regardless of the number of `jdeCacheInit` or `jdeCacheInitMultipleIndex` calls or `jdeCacheTerminate` calls.

Using the Same Cache in Multiple Business Functions or Forms

If the same cache is required for two or more business functions or forms, call `jdeCacheInit` in the first business function or form, and add data to it. After exiting that business function or form, do not call `jdeCacheTerminate` because this removes your cache from memory. Instead, in the subsequent business functions or forms, call `jdeCacheInit` again with the same index and cache name as in the initial call to `jdeCacheInit`. Because the cache was not terminated the first time, JDECACHE looks for a cache with the same name and assigns that to you. Because the cache already has records in it, you do not need to refresh it. You can proceed with normal cache operations on that cache.

If a cache is initialized multiple times across business functions or forms, use `jdeCacheTerminateAll` to terminate all instances of the cache that were initialized. The name of the cache that corresponds to the `HCACHE` passed to this API will be used to determine the cache to destroy. Use this API when you do not want to call `jdeCacheTerminate` for the number of times that `jdeCacheInit` was called. If you move from one form or business function to another when you initialize the same cache across business functions or forms, you will lose your `HCACHE` because it is a local variable. To share the same cache across business functions or forms, do not call `jdeCacheTerminate` when you exit a form or business function if you intend to use the same cache in another form or business function.

JDECACHE Cursors

JDECACHE Cursors (JDECACHE Cursor Manager) is a component of JDECACHE that implements a JDECACHE cursor for record retrieval and update. A JDECACHE cursor is a pointer to a record in a user's cache. The record after the record in which the cursor is currently pointing is the next record that will be retrieved from the cache upon calling a cache fetch API.

Opening a JDECACHE Cursor

Manipulating the JDECACHE data is cursor-dependent. Before the JDECACHE data manipulation APIs will work, a cursor must be opened. A cursor must be opened to obtain a cursor handle of the type `HJDECURSOR`, which must, in turn, be passed to all of the JDECACHE data manipulation APIs (with the exception of the `jdeCacheAdd` API). To open the cursor, call the `jdeCacheOpenCursor` API. A call to this API also makes possible the calls to all the data manipulation APIs (except for `jdeCacheAdd`). If you do not open the cursor, these APIs will *not* work. With this call, the cursor opens a JDECACHE dataset, within which it will work. This API opens the dataset, but does not fetch any data. This means that the cache must be initialized by a call to `jdeCacheInit` and populated by a call to `jdeCacheAdd` before a cursor can be opened.

You can obtain multiple cursors to a cache by calling `jdeCacheOpenCursor` and passing different `HJDECURSOR` handles. In a multiple cursor environment, all the cursors are independent of each other.

When you are finished working with the cursor, you must deactivate it or close it by calling the `jdeCacheCloseCursor` API, and passing an `HJDECURSOR` handle that corresponds to the `HJDECURSOR` handle that was passed to the `jdeCacheOpenCursor`. When a cursor is closed, it cannot be used again until it is opened by a call to `jdeCacheOpenCursor`.

See Also

- ❑ *JDECACHE Multiple Cursor Support* in the *Development Tools Guide* for more information about multiple cursors

HJDECURSOR

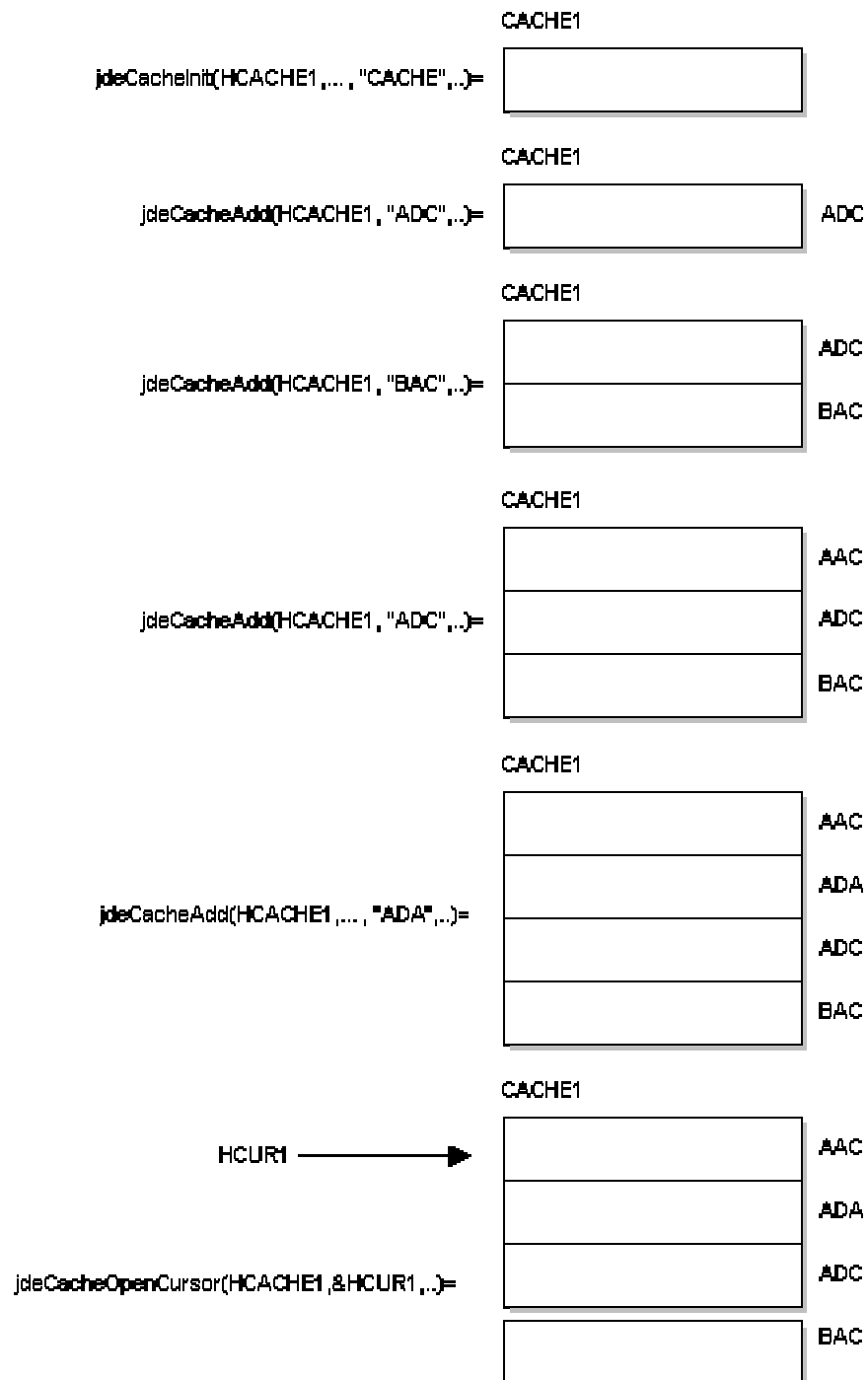
HJDECURSOR is the data type for the cursor handle. It must be passed to every API for JDECACHE data manipulation except jdeCacheAdd.

JDECACHE Dataset

The JDECACHE dataset includes all of the records from the current position of the cursor to the end of the set of sequenced records. Thus, if a cursor is in the middle of the dataset, none of the records in the cache prior to the current position of the cursor is considered part of the dataset. The JDECACHE dataset consists of the cache records sequenced in ascending order of the given index keys. This means that the order in which the records have been placed in JDECACHE is not necessarily the order in which JDECACHE Cursors retrieves them. JDECACHE Cursors retrieves records in a sequential ascending order of the index keys. A forward movement by the cursor reduces the size of the dataset during sequential retrievals. When the cursor advances past the last record in the dataset, a failure is returned.

The following example illustrates the creation of a JDECACHE cache and a JDECACHE dataset:

Example: JDECACHE Cache and Dataset Creation



Cursor-Advancing APIs

Cursor-Advancing JDECACHE Fetch APIs implement the fundamental concepts of a cursor. The cursor-advancing API set consists of APIs that advance the cursor to the next record in the JDECACHE dataset before fetching a record from JDECACHE. The following examples are cursor-advancing fetch APIs:

- `jdeCacheFetch`
- `jdeCacheFetchPosition`

A call to `jdeCacheFetch` first positions the cursor at the next record in the JDECACHE dataset before retrieving it. JDECACHE Cursors also allow calls to position the cursor at a specific record within the dataset. To do this, you call the `jdeCacheFetchPosition` API, which advances the cursor to the record that matches the given key before retrieving it.

You can use a combination of cursor-advancing fetch APIs if you need a sequential fetch of records starting from a certain position. Call `jdeCacheFetchPosition`, passing the key of the record from which you want to start retrieving. This advances the cursor to the desired location in the dataset and retrieves the record. All subsequent calls to `jdeCacheFetch` will fetch records starting from the current cursor position in the data set until the end of the dataset, or until the program stops for another reason.

Non-Cursor-Advancing APIs

Non-cursor-advancing JDECACHE Cursor APIs do not advance the cursor before retrieving a record. Instead, they keep the cursor pointing to the retrieved record. The following examples are non-cursor-advancing fetch APIs:

- `jdeCacheUpdate`
- `jdeCacheDelete`

Updating Records

If you want to update a specific record with a key that you know, call `jdeCacheFetchPosition`, passing the known key, to position the cursor at the location of the record that matches the key. Because the cursor is already pointing to the desired location, call `jdeCacheUpdate`, passing the same `HJDECURSOR` that you used in the call to `jdeCacheFetchPosition`.

If the index key changes, cache re-sorts the records, and the cursor points to the updated location. However, when you call `jdeCacheFetch`, the system retrieves the next record in the updated set. Consequently, the system might not retrieve the correct record because the changed index key caused the order of the records to change.

To update a sequential number of records, make a call to `jdeCacheFetchPosition` to return to the beginning of the sequence, if necessary. Then call `jdeCacheUpdate`, passing the same `HJDECURSOR` that you used in the call to `jdeCacheFetchPosition`. This call updates only the record to which the cursor is pointing. To update the rest of the records in the sequence, call `jdeCacheFetch` repeatedly, passing the same `HJDECURSOR` that you used in the call to `jdeCacheFetchPosition`, until you get to the end of the sequence. A sequential update will not work correctly if you have changed any index key value. However, a sequential update will work correctly if you are updating a value that is not an index key.

Deleting Records

If you want to delete a specific record with a known key, first call `jdeCacheFetchPosition` to point the cursor to the location of the record that matches the key. Next, call `jdeCacheDelete`, to remove the record from cache. Pass `jdeCacheDelete` the same `HJDECURSOR` that you used when you called `jdeCacheFetchPosition`. After deleting a record, use `jdeCacheFetch` to retrieve the record that followed the now-deleted record. This process works only when you call `jdeCacheDelete`.

You can also delete a specific record by calling `jdeCacheDeleteAll` and passing it the full key with the specific record to be deleted. In this case, `jdeCacheFetch` will not work following `jdeCacheDeleteAll`, although you can work around this condition with `jdeCacheFetchPosition` or `jdeCacheResetCursor`.

To delete a sequential set of records, first call `jdeCacheFetchPosition` to point the cursor to the first record in the set or call `jdeCacheDeleteAll` to delete the first record in the set. Then, call `jdeCacheDelete` sequentially. In this case, `jdeCacheFetch` will not work following `jdeCacheDeleteAll`, although you can work around this condition with `jdeCacheFetchPosition` or `jdeCacheResetCursor`.

If you want to delete records that match a partial key, call `jdeCacheDeleteAll` and pass it a partial key. The system deletes all of the records that match the partial key. After you call this API, `jdeCacheFetch` does not work.

The `jdeCacheFetchPosition` API

The `jdeCacheFetchPosition` API searches for a specific record in the dataset; therefore, it requires a specific key. This API can perform full and partial key searches.

Note

If you pass 0 for the number of keys, the system assumes that you want to perform a full key search.

See Also

- *JDECACHE Partial Keys* in the *Development Tools Guide* for a detailed explanation of partial keys

The `jdeCacheFetchPositionByRef` API

The `jdeCacheFetchPositionByRef` API returns the address of a data set. The API finds the one record in cache and returns a reference (pointer) to the data. `jdeCacheFetchPositionByRef` retrieves a single, large block of data that is stored in cache. If the cache is empty or has more than one record, this API fails.

Resetting the Cursor

JDECACHE Cursors supports multiple cursors, as well as an unlimited number of cursor oscillations within the dataset. This means that the cursor can shuttle from beginning to end for an unlimited number of times. The cursor moves forward only. To reset the cursor (move the cursor back to the beginning of the dataset), you must make a call to `jdeCacheResetCursor` API to get a fresh JDECACHE dataset.

You can also reset a cursor to a specific position that is outside of the current dataset by calling the `jdeCacheFetchPosition` API.

See Also

- ❑ *JDECACHE Dataset* in the *Development Tools Guide* for more information about resetting the cursor

Closing a Cursor

When you no longer need the cursor, use a call to `jdeCacheCloseCursor` to close it. This call closes both the dataset and the cursor. Any subsequent call to any JDECACHE API passing the closed `HJDECURSOR` without having called `jdeCacheOpenCursor` will fail.

Although opening a JDECACHE Cursor for a long period of time requires no overhead, to release the memory that it requires, you should close the cursor as soon as you no longer need it.

JDECACHE Multiple Cursor Support

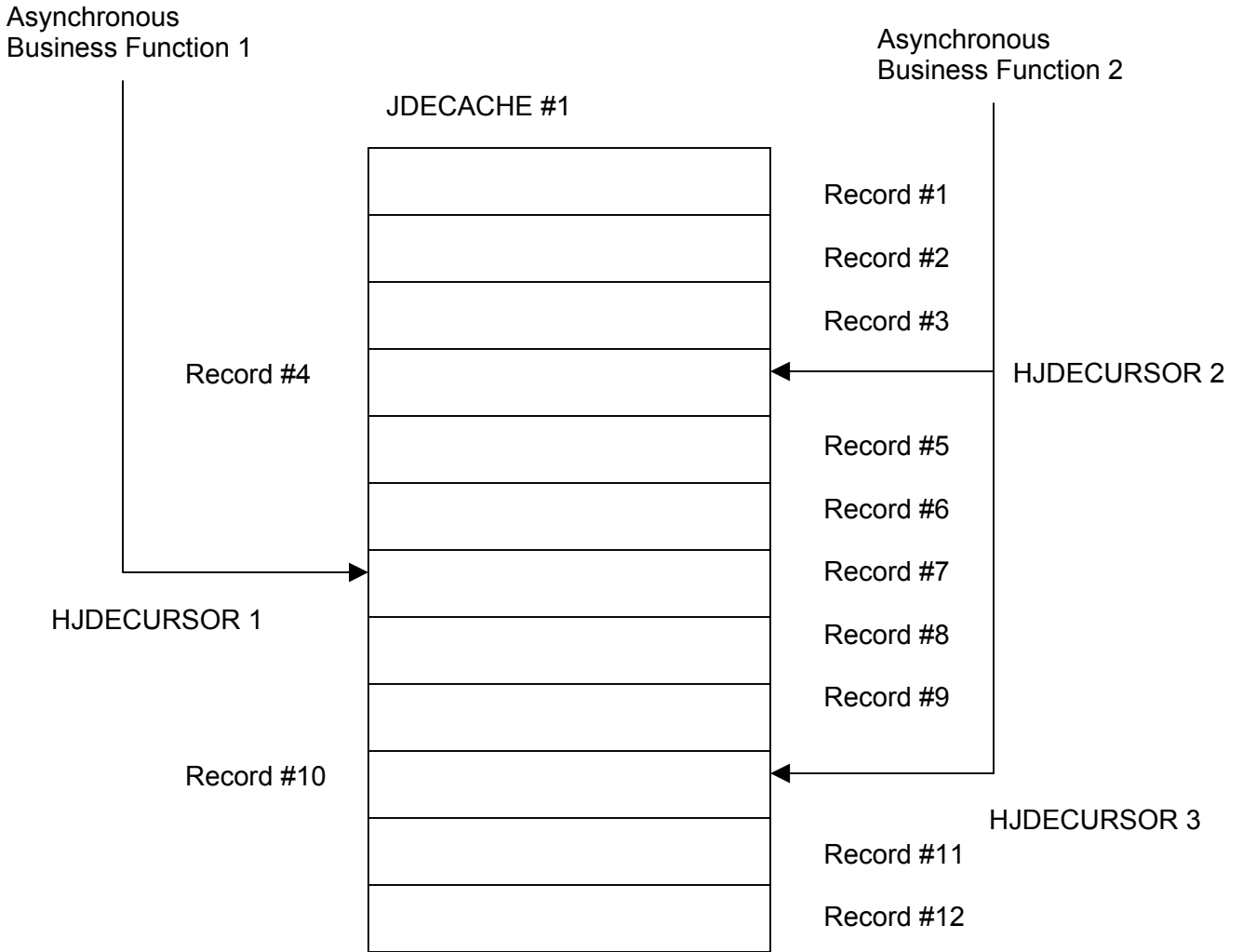
JDECACHE supports multiple open cursors. Each cache allows up to 100 open cursors to access it at the same time.

JDECACHE multiple cursors are designed to allow two or more asynchronously processing business functions to use one cache. Asynchronously processing business functions can open cursors to access the cache with relative positions within the cache that are independent of each other. A cursor movement by one business function does not affect any other open cursor.

Some EnterpriseOne software applications groups restrict the use of multiple cursors. For example, use multiple cursors only if you have a need for them. Additionally, do not use two cursors to point to the same record at the same time unless both cursors are fetching the record.

The following diagram illustrates multiple cursors in JDECACHE:

Multiple Cursors in JDECACHE



JDECACHE Partial Keys

A JDECACHE partial key is a subset of a JDECACHE key that is ordered in the same way as the defined index, beginning with the first key in the defined index. For example, for a defined index of N keys, the partial key is the subset of the keys 1, 2, 3, 4...N-1 in that specific order. The order is critical. Partial key components must appear in the same order as the key components in the index. (The index is passed to `jdeCacheInit`).

For example, suppose that an index is defined as a structure containing the fields in the following order: A, B, C, D, E.

The partial keys that can be synthesized from this index are the following, in order: A, AB, ABC, ABCD.

The previous set is the only set of partial keys that can be synthesized for the defined index: A, B, C, D, E.

How a JDECACHE Partial Key Works

A JDECACHE partial key implements the JDECACHE cursor. When you implement the JDECACHE partial key, consider that the JDECACHE cursor works within a JDECACHE dataset, which comprises the records within the cache *ordered by* the defined index, *the full index*. If you call a `jdeCacheFetchPosition` API and pass the partial key, the JDECACHE cursor activates and points to the first record in the JDECACHE dataset that matches the partial key. If a `jdeCacheFetchPosition` API was called, subsequent calls to `jdeCacheFetch` will fetch all of the records in the dataset that succeed the fetched record *to the end of the dataset*. The cursor does *not* stop on the last record that matches the partial key, but continues on to fetch the next record using the next call to `jdeCacheFetch`, even if it does not match the partial key. When a partial key is sent to `jdeCacheFetchPosition`, it merely indicates from where the JDECACHE begins fetching. Because the records in the JDECACHE dataset are always ordered, the fetch always retrieves all of the records that satisfy the partial key first.

JDECACHE knows that you are passing a partial key because the fourth parameter to `jdeCacheFetchPosition` indicates the number of key fields that are in the key being sent to the API. If the number of key fields is less than the keys that were indicated when `jdeCacheInit` was called, then it is a partial key. Suppose the number of keys is N so that JDECACHE uses the first N key fields to make comparisons in order to achieve the partial key functionality. If `jdeCacheFetchPosition` is called with a number of keys that is greater than the number specified on the call to `jdeCacheInit`, an error is returned.

To delete a partial key, you must make a call to `jdeCacheDeleteAll`. This call deletes all of the records that match the partial key. To indicate to JDECACHE the partial keys that you are using, pass the number of key fields to this API.

Verify that the actual number of key fields in the structure corresponds to the numeric value that describes the number of keys that must be sent to either `jdeCacheFetchPosition` or `jdeCacheDeleteAll`.

JDECACHE Standards

PeopleSoft recommends that you apply the following standards when using JDECACHE.

Cache Business Function Source Name

The cache business function name should follow the standard naming convention for business functions.

Cache Business Function Source Description

The following standards apply to source descriptions for cache business functions:

- The cache business function description follows the business function description standards.
- The first word must be the noun “Cache.”
- The second word must be the verb “Process.”
- For an individual cache function, the words following “Process” should describe the cache. For a common cache function, the words following “Process” should describe the group to which the individual cache functions belong.

Cache Business Function Description

The following standards apply to cache business function descriptions:

- If the source file contains an individual function, the function name should match the source name.
- If the source file contains a group of cache functions, the individual function names should follow the same standards as the Cache Business Function Source Description standards.

Cache Programming Standards

A variety of cache programming standards apply, including the following:

- General standards
- Terminating versus clearing cache
- Cache name
- Defining the cache data structure
- Data structure standard data items
- Cache action code standards
- Group cache business function header file
- Individual cache business function header file

General Standards

Two types of standard cache functions exist: the individual cache function and the group cache function. An individual cache function is a single business function that acts as the file server over a single cache. A group cache function is a single business function that acts as the file server over several predefined caches. The following general standards apply to cache programming:

- Use cache APIs only within a standardized cache function. Using the standard cache function allows you to easily create and maintain code. The standard functions simplify the use of cache.
- Ensure that the cache function contains the standard cache actions. You can also include additional actions against cache that are specific to the cache.
- Use a cache function to initialize the cache, load cache, retrieve records from cache, delete records from cache, modify records, and terminate the cache.

Terminating versus Clearing Cache

You either terminate the cache or clear it, depending on whether the cache needs to remain available. This decision is specific to the design of your application. When you terminate a cache, both the cache index structures and the cache data structures are cleared. When you clear a cache, you delete all records but you do not affect the cache data structure and the cache index structure.

Cache Name

The following standards apply to the cache name:

- The cache name is a maximum of 50 characters.
- Depending on how the cache is used, the cache name should either be the data structure name or the data structure name with the job number appended to it.
- If the cache is to be terminated, the cache name should include the job number.
- When cache is used in master business functions, the cache is not terminated by the application. The job number is stored as a key in the cache. Instead of terminating the cache, all records containing the job number are cleared at the end of the transaction. The cache is not terminated due to the asynchronous processing. In this case, the cache name should contain only the data structure name.

Defining the Cache Data Structure

The cache data structure includes the following:

- Group Cache Function
The data structure for the group cache function is also the layout and index for the individual cache. For the data structure, run typedef, and then paste it into the business function header file.
- Individual Cache Function
The cache layout and cache index for an individual cache function should be defined in the business function header file. The caches layout and cache index should always be the same.

Data Structure Standard Data Items

The following data items are standard to every cache business function. These data items should be the first data items in the data structure.

- NACTN - Cache action code
- NKEYS - Number of keys
- CURSOR - Cache cursor
- DTAI - Error Message ID
- JOBS - Job number (optional)

Additional data items that pass the information stored in cache should follow the standard data items. Preferably, the key fields are listed first, in order of the cache index.

Standard Cache Action Codes

Use the following predefined standard cache action codes.

CACHE_GET

- Opens cursor.
- Fetches a single record from the cache using the jdeCacheFetch API.

Fetch can be with either a full or partial key. The number of keys used is passed to the function through the standard data items for the business function data structure.

- Does not require cache cursor to perform the fetch, and does not return the cache cursor after the record is fetched.
- Returns success or failure.

CACHE_ADD

- Inserts a record into cache using the `jdeCacheAdd` API.
- Must have a full key defined, and must pass values for all key fields.
- Does not allow duplicate records in cache.
- Does not require cache cursor to add a record to cache, and does not return the cache cursor after the record is added.
- Returns success or failure.

CACHE_UPDATE

- Opens cursor.
- Fetches the record for update from cache using the `jdeCacheFetchPosition` API (optional).
- Updates the fetched record using the `jdeCacheUpdate` API.
- Must have a full key, and must pass values for all key fields.
- Does not require cache cursor to update a record in cache, and does not return the cache cursor.
- Returns success or failure.

CACHE_ADD_UPDATE

- Deletes an existing record and replaces it with the new record. If a record does not exist, adds it.
- Fetches the record using the `jdeCacheFetch` API. If a record is found, deletes it using the `jdeCacheDelete` API. Adds the new record with the values passed through the business function data structure using the `jdeCacheAdd` API.
- Requires that a full key be defined, and must pass all values for the key fields.
- Does not require the cache cursor, and does not return the cache cursor.
- Returns success or failure on the new record being added.

CACHE_DELETE

- Deletes one record, a set of records, or all records from cache, depending on the number of keys passed.
- Deletes a single record from cache using `jdeCacheDelete` API if the number of keys is equal to the total number of keys in the index.
- Can be used to delete a set of records from cache by defining a partial key. Records are deleted using `jdeCacheDelete` API.

- Deletes all records from cache if the number of keys is set to zero. Records are deleted using `jdeCacheDeleteAll` API.
- Does not require a cache cursor to delete a record from cache, and does not return the cache cursor.
- Returns success or failure.

CACHE_GET_NEXT

- Opens the cursor, and then fetches the first record using `jdeCacheFetchPosition`, if the cache cursor is empty.
- Fetches the next record if the cursor contains a value.
- Passes a full or partial key. The number of keys must remain constant.
- Compares the value of the defined number of keys in the previous record with the next record to determine the end of file.
- Closes the cursor when a key match fails and frees the data pointer.

CACHE_TERMINATE

Terminates the cache using the `jdeCacheTerminate` API. The cache terminates only when the number of users of the cache equals zero.

CACHE_TERMINATE_ALL

- Terminates the cache using the `jdeCacheTerminateAll` API.
- Frees the data pointer.
- Terminates the cache regardless of the number of users accessing the cache.

CACHE_CLOSE_CURSOR

Closes the cursor using the `jdeCacheCloseCusor` API.

Additional Features

PeopleSoft provides additional development tools and features that you can use to enhance your applications. These additional features include the following:

- Processing options
- Transaction processing
- Record locking
- Currency
- Tips of the day

Processing Options

Processing options control how an interactive or batch application processes data. You can use processing options to change the way in which an application or a report appears or behaves. You can attach unique processing options to different versions of the same application, which allows you to change the behavior of an application without creating a new application. In addition, you can use processing options to do the following:

- Control the path that a user can use to navigate through a system.
- Set up default values.
- Customize an application for different companies or different users.
- Control the format of forms and reports.
- Control page breaks and totaling for reports.
- Specify the default version of a related application or batch process.

You can define processing options for an application that automatically appear at runtime.

In addition, you might need to create a processing option version. The procedures for creating a processing option version are similar to those for creating an interactive version.

Processing Options Templates

A processing options template contains one or more processing options. Each processing option appears on a row within the template and is defined by the following three variables:

Tab Title	A title that categorizes processing options. This text appears on the tab for a processing option.
Comment	Optional text that appears on the form with the processing options. Each comment takes the place of a processing option on the page. Adding a comment eliminates space for a processing option.

Data Item An entry in the data dictionary. Every processing option must be a data item in the data dictionary. You specify data items for which you want to assign default values, such as cost center ranges and dates.

At runtime, a processing option template displays a set of tabs within an area called a page. Each tab represents a category of processing options. When you click the tab, the page changes to show the set of processing options for that category.

Caution

Changes to processing option text can conflict with changes to processing option templates. Template changes do not take effect until another package is built, but text changes occur immediately.

The following steps describe how to create and implement processing options:

1. Create processing options by building a list of parameters called a template.
2. Attach this template to an application and create event rules so that the application uses these values.
3. Create versions of the application.
4. Specify how the processing options are handled at runtime.

At runtime, depending on how you set up the application, one of the following events occurs:

- The processing options appear, and the user can choose to supply values to processing options.
- A version list appears from which the user can choose a version.
- The system runs the application with a predefined set of options.

When working with processing option design aid (PODA), all processing option template information is stored in the POTEXT TAM until you check it in. When you check in the processing option template, it is moved from POTEXT TAM to the Processing Option Text table (F98306). Data values for processing options are stored in the Versions List table (F983051). For batch versions, the Versions List table has an identifier that points to specifications for overrides (report overrides, data sequencing, data selection, or override location).

Each version of an application can be associated with a list of processing option values. The processing options that are specified at the time that an interactive or batch application is launched are those that the application uses when it runs. This sequence prevents users from changing processing options between the time that a batch application is submitted and the time it actually runs. Processing options can also be used for a specific running of an application. These processing options are not permanently stored in the F983051 table, and they are used only for that specific running.

Defining a Processing Options Data Structure (Template)

You can create a processing options data structure (template) that lists the values for data items that are passed to the application at runtime. Any changes that you make to the template reside on your workstation until you check in the template. This ensures that current users of the existing template are not immediately affected by your changes. After you check in your changes, the next Just-In-Time-Install (JITI) replicates the changes to the other users.

Prerequisite

- ❑ Create a processing options data structure. See *Creating a Processing Options Data Structure* in the *Development Tools Guide*

► To define a processing options data structure (template)

1. On Object Management Workbench, check out the processing options data structure with which you want to work.
2. Ensure that the data structure is highlighted, and then click the Design button in the center column.
3. On Processing Option Design, click the Design Tools tab, and then click *Start the Processing Option Design Aid*.

The Processing Options Design tool launches. The area on the left of the form displays how the processing option will look to the user.

4. Locate the data items that you need for your processing options with the Data Dictionary Browser.
5. Use one of the following methods to choose the data items that you want to add to your processing options:
 - Double-click the item in the Data Dictionary Browser. The item appears in the left side of the form under the tab.
 - Drag the item from the Data Dictionary Browser to the position where you want it in the structure members.
6. Click an item to edit it.
 - You can use the hatching around the control to reposition it.
 - You can choose text, and then delete or overwrite it.

Processing Options Design automatically adjusts the size and position of data items to fit the width of the tab.

7. Choose the text button (A) to add comments.
8. Choose an object in the area on the left side of the form, and choose Properties from the View menu.

If you are on a data item, you can view its properties and change the item name if necessary. The item name should be unique.

You can click the Help Override Data Item tab to add an alternate szDict from which to get the help.

9. Right-click the processing option, and then choose Properties from the menu.
10. On JDE.DataItem Properties, click the Help Override Data Item tab, and then complete the following field:

- Data Item Help Override Name

Note

When you name Help Override Data Items, you should use the naming guidelines as defined in the Development Guidelines for Application Design.

11. Click OK.
12. To view tab properties, click the tab and choose Properties from the View menu.

You can also right-click a tab and choose Current Tab Properties from the menu that appears.

If you are on a tab item, you can enter a short and long name for the tab.

Use the Help File Name field to add the name of the help file for the tab.
13. To add a new tab, choose New Tab from the File menu.

You can also right-click on an existing tab and choose New Tab from the menu that appears.

EnterpriseOne Processing Option Naming Standards

You should follow EnterpriseOne naming standards whenever possible unless you have a strong business case for doing otherwise. Following the naming standards ensures a consistent approach to programming.

Processing Option Data Structure

The Object Librarian name for a data structure can be a maximum of 10 characters—only 9 if you begin the name with a T—and is formatted as follows:

Txxxxxyyyy

Where:

T = Processing option data structure

xxxxxyyyy = The program number for the application or report

For example, the processing option data structure name for the P0101 application is T0101.

Tab Title

Use the following guidelines when you define a tab title for a processing option:

- Avoid abbreviations.
- For future processing options, indicate that they are currently unavailable by entering the word FUTURE. If the entire tab is unavailable, enter FUTURE behind the extended

description for the tab. If a single processing option is unavailable, place FUTURE behind the data item description.

- Ensure that each tab exists only one time and that it is not divided into multiple tabs. For example, use Process instead of Process 1, Process 2.
- Include the application name, such as P4310, in the text when referencing versions that are to be used. The Version tab should always begin with the comment block "Enter the version to be used for each program. If left blank, ZJDE0001 will be used."
- Use application-specific tabs sparingly and only when no other categories are appropriate. To allow for increased length of text when the text is translated, the name of an application-specific tab should be no longer than 10 characters in English.
- Use one of the eight standard tab titles. Along with the extended description and processing options for each, they are as follows:
 - Display: Options that determine whether specific fields appear or which format of a form appears on entry.
 - Defaults: Options that assign default values to specific fields.
 - Edits: Options that indicate whether the system performs data validation for specific fields.
 - Process: Options that control the process flow of the application.

Application-specific tabs are:

- Currency: Options that are specific to currency.
- Categories: Options that assign default category codes.
- Print: Options that control the output of a report.
- Versions: Options that specify which versions the system runs of applications that are called from this application.

Comment

- When you enter a comment for a processing option, use the following guidelines:

Note

When several processing options are grouped together, you can choose to number the processing options or the comments. Choose whatever works best for the situation.

- Number every option on a tab. Use sequential numbering, starting at 1, for each tab.
- Use nouns, such as Customer Master, to describe the processing option. The action required is defined in the glossary for that processing option.
- Add the word "Required" to the end of the processing option if a processing option is required.
- Use a comment block when multiple processing options refer to the same topic. The comment block is a title for the logical group of processing options.

Data Item

When choosing a data item for a processing option, use the following guidelines:

- When necessary, change the name of the data item to be descriptive.
- When renaming the data item element, the field element should comply with the naming standards for event rule variables, with the alias appended, such as szCategoryCode3_CT03.
- Use a relevant data item when the data dictionary glossary applies. The user can display the glossary from the processing options. Do not use generic work fields, such as EV01.

Language Considerations for Processing Options

You can change a processing option template to incorporate language features.

► To change a template for text translation

From the System Administration Tools menu (GH9011), choose Processing Options Text Translation.

1. On Work With PO Text Translations, complete the following fields and click Find:

- Template Name
- To Language

Work with PO Text Translations displays processing option text for the processing option template and language that you specify.

2. Choose a row with the text type that you want to change and click Select.

Text types include tabs, items, and comments.

3. On PO Text Translation, enter the new text.

► To add a template with translated text

When you add a new processing option template for an application that is language-enabled, complete the following tasks:

1. Create the application.
2. Create the processing option template for the base language.
3. Add the language text.

Attaching a Processing Options Template

You must attach a processing options template (data structure) to an application to enable processing options at runtime. You can use processing options to set up default values, control formats, control report breaks, control totaling, and control how a report processes data. A processing options template has the following characteristics:

- It exists as a separate object.
- It can be attached to multiple applications.

When you attach a processing options template, if any of the processing options are designed to process on a certain event, you must attach event rule logic to enable those processing options.

► **To attach a processing options template**

1. On Application Design Aid, choose Design from the Form menu.
2. From the Application menu, choose Select Processing Options.
3. On Select Processing Option Template, choose the processing option template that you want to use and click OK.

Caution

The versions of an application and the event rules attached to it depend on the block of data passed to the application by way of a specific processing option template. If you disconnect that template from the application or connect a different template to the application, the application might not run properly.

To change the processing option template, first remove all existing versions of the application. Then, examine all application objects for event rules to ensure that the data that they need will still be available after you change or remove the template.

Transaction Processing

A transaction is a logical unit of work (comprised of one or more SQL statements) performed on the database to complete a common task and maintain data consistency. Transaction statements are closely related and perform interdependent actions. Each statement performs part of the task, but all of them are required for the complete task.

Transaction Processing ensures that related data is added to or deleted from the database simultaneously, thus preserving data integrity in your application. In transaction processing, data is not written to the database until a commit command is issued. When this happens, data is permanently written to the database.

For example, if a transaction comprises database operations to update two database tables, either all updates will be made to both tables, or no updates will be made to either table. This condition guarantees that the data remains in a consistent state and the integrity of the data is maintained.

You see a consistent view of the database during a transaction. You do not see changes from other users during a transaction.

Transaction Processing ensures that transactions are:

- Atomic - Either all database changes for an entire transaction are completed or none of the changes are completed.
- Consistent - Database changes transform from one consistent database state to another.

- Isolated - Transactions from concurrent applications do not interfere with each other. The updates from a transaction are not visible to other transactions that execute concurrently until the transaction commits.
- Durable - Complete database operations are permanently written to the database.

Commits and Rollbacks

The scope of a transaction is defined by the beginning and the end of the transaction. The end of a transaction occurs when the transaction is committed or rolled back. If neither a commit nor a rollback of a transaction occurs, the transaction rolls back when you exit the system.

Transaction processing uses commits to control database operations. Commits are commands to the database. Transactions can be automatically or manually committed. For automatic commits, database changes are written permanently to the database (committed) as they are executed. For manual commits, database changes are written permanently to the database only when either a commit or a rollback occurs.

Commit

A commit is an explicit command to the database to permanently store the results of operations performed by a statement. This event successfully ends a transaction.

Two-Phase Commit (Manual Commit Mode)

A two-phase commit coordinates a distributed transaction. It occurs only when at least one update statement has been executed to two separate data sources in the same transaction.

Rollback

A rollback is an explicit command to the database to cancel the results of operations performed by a statement. This event indicates that a transaction ended unsuccessfully.

Any failure of an insert, update, or delete within a transaction boundary causes all record activity within that transaction to roll back. If no failures have occurred at the end of the transaction, a commit is done, and the records become available to other processes.

In the case of a catastrophic failure (such as due to network problems), the DBMS performs an automatic rollback. Likewise, if the user clicks Cancel on a form, a rollback command is issued through a system function.

Transaction Processing

An EnterpriseOne software transaction is a logical unit of work (comprised of one or more SQL statements) performed on any number of databases. A single-statement transaction consists of one statement; a multiple-statement transaction consists of more than one statement.

You can construct a transaction within an EnterpriseOne application to group multiple database operations. The application can then request the database management system to buffer the database operations until the application executes a specific command to perform the updates requested within the transaction. Database operations that are not part of a transaction update the database immediately.

If an application has transaction processing turned on, you cannot see updated records until an update has been committed. Only processes within that transaction can access records in the transaction until the transaction is complete.

The Application Design tool allows you to enable an application for transaction processing and to define which database operations comprise a transaction. Not all transactions or applications must be enabled. Enable transaction or applications appropriately, according to your database configuration.

If transaction processing is turned on for database operations for tables that reside in DB2, then those tables must be journaled. Journaling can decrease performance because of the additional processing required. Contact your DB2 administrator if you have problems with this process.

General messages and errors for transaction processing are written to the jde.log or jdedebug.log.

Data Interdependence

Data interdependence refers to the data elements that make a transaction complete. For example, a voucher has records in both the Accounts Payable Ledger table (F0411) and the Account Ledger table (F0911). Because no data interdependence exists between the two tables, the transaction is incomplete when data exists in one table and not the other.

Transaction Boundaries

Data interdependence is defined by a transaction boundary. A transaction boundary encompasses all of the data elements that comprise a transaction. A transaction boundary might include only the data elements on a single form. When a transaction includes data from another form, the transaction boundary must be extended to include the data on that form.

Transaction Processing Scenarios

The typical flow for a transaction is as follows:

1. An application starts and calls the runtime engine.
2. The runtime engine initializes the transaction.
3. The runtime engine opens a view.
4. The runtime engine performs database operations.
5. The runtime engine commits database operations.

To include two connected forms in the same transaction boundary, you must activate transaction processing for the parent form and designate *Include in parent* on interconnect to the second form. You do not need to activate transaction processing for the second form because your choice on the interconnect form overrides your choice on the called form.

The following table outlines the relationship between two forms and the boundaries that exist in each scenario. Transaction boundaries are defined through form interconnections and business function interconnections. In the example below, the OK button on Form1 invokes Form2. You can change the transaction boundaries by specifying TP On or TP Off. The table explains what happens when you define your transaction boundary in various ways.

Scenario		TP On	TP Off	Form, BSFN Interconnect, Table I/O	Comments
A	Form1		X		All forms use Auto Commit.
	Form2		X		
B	Form1		X	X	Because neither form uses Manual Commit, the Include in Parent flag on Form Interconnect Properties is ignored.
	Form2		X		All forms use Auto Commit.
C	Form1	X			Form1 (parent) uses Manual Commit mode, and Form2 (child) uses Auto Commit.
	Form2		X		Because the Include in Parent flag is Off, the transaction boundary does not extend to include Form2 (child).
D	Form1	X		X	Even though the transaction processing flag is Off for Form2 (child), the Include in Parent flag is On.
	Form2		X		The transaction boundary extends to include Form2 (child).
E	Form1		X		Because the Include in Parent flag is Off, Form1 (parent) and Form2 (child) operate as independent entities.
	Form2	X			Form1 operates in Auto Commit mode and Form2 operates in Manual Commit mode.
F	Form1		X	X	An atypical case. Because transaction processing is Off for Form1 (parent), the transaction boundary does not extend to the child, even though the Include in Parent flag is On for Form2 (child).
	Form2	X			Form2 (child) is in Manual Commit mode and the interconnect is ignored.
G	Form1	X			Transaction processing is On for both forms.
	Form2	X			Because the Include in Parent flag is Off, each form is a transaction boundary and a commit is issued for each.
H	Form1	X		X	Transaction processing is On for both forms. However, because the Include in Parent flag is On, the transaction processing on Form2 is ignored.
	Form2	X			The transaction boundary encompasses both forms. Form2 is a child of Form1.

Transaction Processing and Business Functions

An application or batch process establishes the primary transaction boundary. If a business function calls another business function, the database operations in the function that is being called are still grouped within the boundaries of the parent application.

Master business functions should not define their own boundaries. You might require two or more master business functions to create one logical transaction; in such cases, the calling application should define the boundaries.

If your application calls several business functions, and you need to include the business functions in the transaction boundary, you must enable transaction processing. Should a failure occur and you need to roll back database operations for the business function, you must designate Include in Transaction on the business function interconnect.

Note

When you use business functions within a transaction, you must be careful not to cause a deadlock. If you split the logic for manipulating a table between two functions, you might cause a deadlock if you include one function in the transaction but not the other. If a business function that selects records for information also updates or inserts data in other tables, you might want to split the business function.

Transaction Processing in Remote Business Functions

In a transaction-enabled application, if a server business function has modified a record, and a client business function outside the transaction attempts to access the record, the client function is locked out until the server business function has committed the data. Until the data is committed, the client application cannot access database changes performed by server-side business functions. If a server business function fails to commit or a user cancels a transaction on a server business function, the transactions for the business function roll back. The servers and client must all commit the transaction, or the transaction rolls back on the servers and client.

Transaction Processing System Functions

Several transaction processing system functions are available. You might need to use system functions for additional transaction processing functionality.

For example, assume a scenario with two forms—FormA and FormB—with transaction processing enabled for FormA. Furthermore, assume that FormA calls FormB with the Include in Parent option on for the *Post OK Button is Clicked* event. Because FormB inherits the transaction boundaries for FormA, if a user cancels an event on FormB, the following occurs:

- The entries for FormB will not be written.
- Control is returned to FormA.
- The entries for FormA are written and committed.

In this scenario, you can prevent commitment of the entries for FormA by using the Rollback Transaction system function.

You can use the following system functions to define transaction boundaries in a batch process:

- *Begin Transaction* to define where the transaction begins
- *Commit Transaction* to define where the transaction ends
- *Rollback Transaction* to rollback a transaction

Working with Transaction Processing

Transaction processing is available for the following form types:

- Fix/inspect
- Header detail
- Headerless detail

Transaction processing is only available during OK processing for the following events:

- OK Button Clicked
- OK Post Button Clicked
- Add Record to DB - Before
- Add Record to DB - After
- Update Record to DB - Before
- Update Record to DB - After
- Add Grid Rec to DB - Before
- Add Grid Rec to DB - After
- All Grid Recs Added to DB
- Update Grid Rec to DB - Before
- Update Grid Rec to DB - After
- All Grid Recs Updated to DB
- Delete Grid Rec from DB - Before
- Delete Grid Rec from DB - After
- All Grid Recs Deleted from DB

Actions that occur outside of these events are not within the transaction boundary.

Defining Transaction Processing for a Form

To define transaction processing for a form, you must specify the Transaction option on Form Properties in Form Design. This requirement means that all data for the form is committed to the database at the same time.

If a transaction includes a single form, then this is the only action that is required because the form itself is the transaction boundary. However, if the transaction includes data from another form, then you must extend the boundary to the applicable form through a form interconnection.

Note

You can also extend transaction boundaries using business functions or table I/O.

See Also

- ❑ *Extending a Transaction Boundary* in the *Development Tools Guide* for more information about transaction boundaries

► To define transaction processing for a form

1. On Form Design, double-click the form for which you want to access Form Properties.
2. Click the following Style option:
 - Transaction Processing

Transaction Processing for reports occurs at the section level.

Extending a Transaction Boundary

You can extend the transaction boundary from one form to another form by setting up a parent/child relationship between the forms. To extend the boundary, enable the Transaction Processing flag through a form interconnection in Event Rules Design.

Prerequisite

- ❑ In Form Design, define form properties so that each form within the transaction boundary includes transaction processing. See *Defining Transaction Processing for a Form* in the *Development Tools Guide*.

Extending a Transaction Boundary between Forms

If a parent form uses manual commit, the form to which you connect it must also use manual commit.

► To extend the transaction boundary between forms

1. Open Form Design Aid.
2. Choose the parent form with which you are working.
3. From the Form menu, choose Menu/Toolbar Exits.
4. Choose the OK row and click the Event Rules button.
5. From Event Rules Design, choose the Button Clicked event, and click the Form Interconnect button.
6. On Work with Applications, choose the application that you want to use.
7. On Work with Forms, choose the form that you want to include in the transaction boundary.
8. On Work with Versions, choose the version of the application that you want to use.
9. On Form Interconnect, click the following Transaction Processing option and click OK:
 - Include in Transaction

Extending a Transaction Boundary By Using Business Functions

You can include a business function in a transaction boundary. If the parent form uses automatic commit, the business function to which you extend the transaction boundary also uses automatic commit. Any business function that is not marked as Include in Transaction uses auto-commit. Business functions that process asynchronously can also be included in a transaction.

► To extend a transaction boundary by using business functions

1. Open Form Design Aid.
2. Choose the parent form with which you are working.
3. From the Form menu, choose Menu/Toolbar Exits.
4. Choose the OK row and click the Event Rules button.
5. From Event Rules Design, choose the *Button Clicked* event, and click the Business Functions button.
6. From Business Function Search, choose the business function that you want to include in the transaction boundary.
7. On Business Function, click the following Transaction Processing option:
 - Include in Transaction

Business Functions marked for both Asynchronous and Include in Transaction are included in the transaction boundary.

Extending a Transaction Boundary By Using Table I/O

Transaction processing is available only for the Open Table operation in Table I/O. The opening of a table determines whether interaction with that table will be manual commit (part of a transaction) or automatic commit. Any Open Table operation not marked as Include in Transaction uses automatic commit.

► To extend a transaction boundary by using table I/O

1. Open Form Design Aid.
2. Choose the parent form with which you are working.
3. From the Form menu, choose Menu/Toolbar Exits.
4. Choose the OK row and click the Event Rules button.
5. From Event Rules Design, choose the *Button Clicked* event, and click the Table I/O button.
6. On Insert TableIO Operation, choose the Open option under Advanced Operations, and then click Next.
7. On Data Source, click Advanced Options.
8. On Advanced Options, choose the *Include in Transaction* option, and then click OK.
9. On Data Source, click Finish.

The Open operation appears in your event rules.

Defining Transaction Processing for a Report

In addition to interactive transaction processing, EnterpriseOne software also provides transaction processing for reports and batch processes. To enable transaction processing for a batch process, click the Advanced tab for report properties and choose Transaction Processing. Then, use the Transaction Processing system functions to define the beginning and ending boundaries of your transactions. You can also extend your transaction boundaries to include business functions and table I/O.

Setting the jde.ini for Lock Manager

You must modify the enterprise server and workstation jde.ini files to enable transaction processing. For each EnterpriseOne workstation, you must enable transaction processing by changing settings in the workstation jde.ini file in the [LOCK MANAGER] section. You should make these changes on the deployment server to the resident jde.ini file that is delivered to workstations through package deployment, and then deploy a package with the changed jde.ini file.

The following tasks describe how enter the settings for [LOCK MANAGER].

► To enter [LOCK MANAGER] settings for the server

1. Locate the jde.ini file on the enterprise server.
2. Using an ASCII editor such as Notepad, view the jde.ini file to ensure the accuracy of the following settings:

```
[LOCK MANAGER]
Server=server name
AvailableService=available server service
RequestedService=client service request
```

The following table explains the variables:

Setting	Value
Server	This setting specifies the name of the lock manager server to be used to process records. For example, a server name might be <code>intelnta</code>. If the client is used as a server, such as in cases where batch applications are running on the workstation, this setting must match the same entry in the [LOCK MANAGER] section of the jde.ini file on the workstation.
AvailableService	This setting indicates the available service of the server. Valid values are: • TS: Time stamp service is available • NONE: No service is available This setting applies only to servers.
RequestedService	This setting indicates the type of service that the client requests from the server. Valid values are: • TS: Time stamp service is requested • NONE: No service is requested

Caution

Enable transaction processing on the server before you enable it on the workstation. If you try to set up the workstation jde.ini file before you set up the server jde.ini, you could be requesting a service on the server that is not yet available, which generates an error.

► To enter [LOCK MANAGER] settings for the workstation

1. Locate your workstation jde.ini file.
2. Using an ASCII editor such as Notepad, view the jde.ini file to ensure the accuracy of the following settings:

<pre>[LOCK MANAGER] Server=server name RequestedService=client service request</pre>

The following table explains the variables:

Setting	Value
Server	This setting specifies the name of the lock manager server to be used to process records. For example, a server name might be <code>intelnta</code>. If the client is used as a server, such as in cases where batch applications are running on the workstation, this setting must match the same entry in the [LOCK MANAGER] section of the jde.ini file on the workstation.
RequestedService	This setting indicates the type of service that the client requests from the server. Valid values are: • TS: Time stamp service is requested • NONE: No service is requested

Record Locking

EnterpriseOne does not implement any record-locking techniques. It relies on the native locking strategy of the vendor database management system. This process improves performance by reducing duplication of effort.

In specific situations, the vendor database does not automatically lock as needed. In these situations, you can use explicit instructions to EnterpriseOne to control record locking. For example, you can use record locking to ensure the integrity of the Next Numbers feature.

Understanding Record Locking

You can lock EnterpriseOne records using one of the following methods:

- **Optimistic locking**
Use optimistic locking (sometimes referred to as record change detection) to prevent a user from updating a record if it has changed between the time the user inquired on the record and when the user updates the record.
- **Pessimistic locking**
Use pessimistic locking to prevent attempts to update the same record at the same time. The record is locked before it is updated.

Optimistic Locking

You can turn on optimistic locking within the workstation jde.ini file. This type of database locking prevents a user from updating a record that changes during the time the user is inquiring about it. If the record has changed, the user must select the record again and then make the change. This feature is available for business functions, table I/O, and Named Event Rules.

For example, assume that two users are working in the Address Book application. The following table illustrates the optimistic locking process:

Time	Action
10:00	User A selects Address Book record 1001 to inspect it.
10:05	User B selects Address Book record 1001 to inspect it. Both users now have Address Book record 1001 open.
10:10	User B updates a field in Address Book record 1001 and clicks OK. EnterpriseOne updates Address Book record 1001 with the information entered by User B.
10:15	User A updates a field in Address Book record 1001 and clicks OK. EnterpriseOne does not update Address Book record 1001, and the system displays a message informing User A that the record has changed during the time that User A was viewing it. For User A to change the record, User A must reselect it and then update it.

When the system detects that a record change has occurred, it displays a message indicating that the record has been changed since it was retrieved.

Pessimistic Locking

Pessimistic locking is sometimes referred to as record locking. You can use pessimistic locking to prevent multiple users or applications from updating the same record at the same time. For example, suppose a user enters a transaction that uses Next Numbers. When the user clicks OK, the Next Numbers feature selects the appropriate Next Numbers record, verifies that this number is not already in the transaction file, and then updates the Next Numbers record by incrementing the number. If another process tries to access the same Next Numbers record before the first process has successfully

updated the record, the Next Numbers function waits until the record is unlocked and then completes the second process.

Pessimistic locking in EnterpriseOne is implemented by calling published JDEBase APIs. When you use pessimistic locking, you should consider the time required to select and update a record because the record is locked until the update is complete. Transaction processing uses a special set of locking APIs. A locked record might or might not be part of a transaction. Record locking APIs are independent of the transaction and its boundaries. They always lock, regardless of whether you are in manual or auto commit mode.

Records that are updated using pessimistic locking APIs (such as JDB_FetchForUpdate or JDB_UpdateCurrent) within a transaction boundary are locked from the time the record is selected for update until the commit or rollback occurs. Records within the transaction boundary that are updated without using pessimistic locking APIs are locked from the time of the update until the commit or rollback occurs. This is also true if you use a business function to define and activate transaction processing.

Using Pessimistic Locking Within a Transaction Boundary

You might need to use pessimistic locking in conjunction with transaction processing. For example, if you want the system to lock records between the read operation and the update, you must use pessimistic locking.

Business Functions and Pessimistic Locking

You might want to use pessimistic locking in a business function if the business function updates a table. The table being updated should have a high potential for record contention with another user or job. Remember that you should lock records for as short a time as possible. Ensure that the select or fetch for an update occurs as closely to the update as possible.

Currency

Enterprises that do business internationally have additional accounting needs and added complexity. This complexity arises from doing business in different currencies and having to follow different reporting and accounting requirements. Some fundamental requirements for an international enterprise include:

- Conversion of foreign currencies to the local currency
- Conversion of the different local currencies into one currency for reporting and comparisons
- Adhering to regulations defined in the countries of operation
- Revaluation of currencies due to fluctuation in exchange rates

Currency Implementation

EnterpriseOne currency implementation includes the following features:

- Currency retrieval is accomplished through database triggers and table event rules.
- Currency retrieval logic is handled in business functions.
- System APIs assist you in accessing cached tables.

Advantages

EnterpriseOne allows developers to control currency retrieval. Allowing developers, instead of the system, to control currency, provides greater flexibility and easier maintenance. Some of the advantages in allowing developers to control currency are:

- Additional currency tables do not require changes to system modules. Only new business functions need to be added.
- Business logic is captured in business functions, rather than in a system module that assumes knowledge of business logic.
- Table event rules allow you to attach currency retrieval logic at the table object level.
- Table event rules are triggered by table events instead of application events.
- Any application that uses the table that has currency business functions attached to it receives the same logic, so you do not need to modify each application.
- No hard-coded logic is embedded in the runtime engine.

Working with Currency

When identified amounts are written to or retrieved from a database, or when they are used in calculations during processing, proper decimal placement is extremely important. Currency implementation is needed to adjust decimal placement on Math_Numeric currency fields according to a specified currency. Common applications of currency implementation include conversion of currency amounts and revaluation of currency due to fluctuations in exchange rates.

Implementing currency involves the following steps:

- Performing currency setup.
- Creating a business function that contains logic to retrieve currency information. Currency business functions are known as currency triggers.
- Attaching a currency trigger to the *Currency Conversion* event in Table Event Rules (TER).
- Designing the TER functions through Event Rules Design. The system then converts the event rules to C and compiles them into a consolidated DLL through the Object Configuration Manager (OCM) Application.
- Modifying applications as necessary.

The JDB APIs then calls the appropriate TER function when the *Currency Conversion* event is triggered.

Understanding the Build Triggers Option

The Build Triggers option performs the following steps:

- Converts event rules to C source code.
This creates the files *OBNM.c* and *OBNM.hxx* (where *OBNM* is the Object Name). The source file will contain one function per TER event.

For example, if you are working with the Accounts Payable Ledger table (F0411), the Build Triggers option creates a C source member called F0411.c. You can browse through the C code and ensure that all of the parameters are set up correctly. The system generates an error log if an error occurs during the ER-to-C conversion. The error table is called eF0411.log.

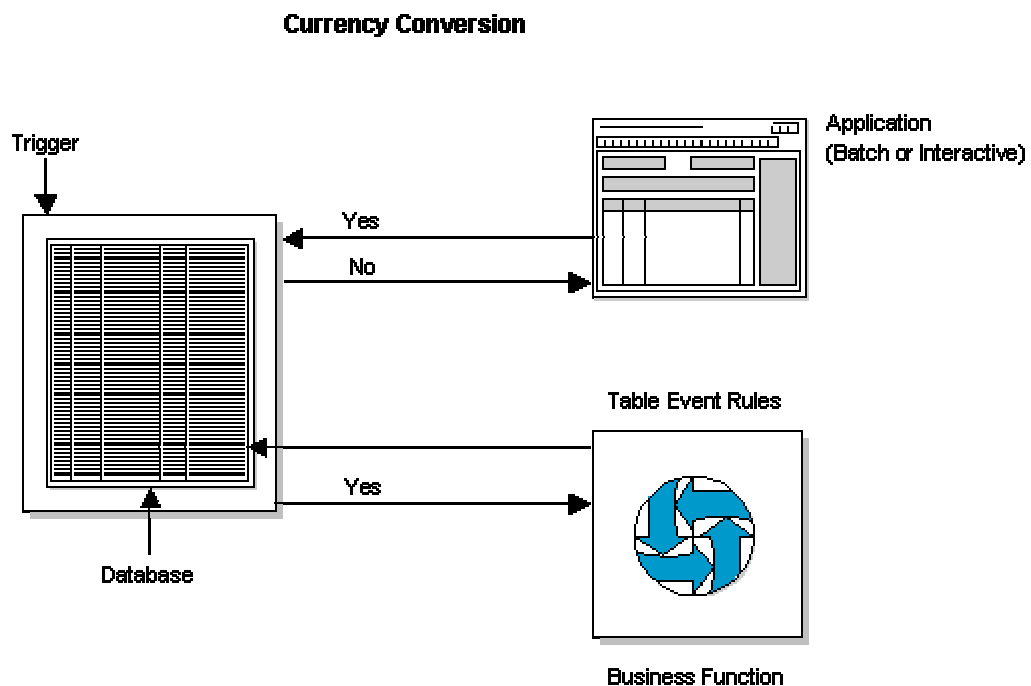
- Compiles the new functions and adds them to JDBTRIG.DLL. This is the consolidated DLL that contains TER functions.

Understanding How Table Event Rules Work with Currency Processing

The *Currency Conversion* event runs if currency processing is enabled.

Table triggers for currency run after the record is fetched and before the record is added to the database.

The following graphic illustrates the currency conversion process:



On FETCH:	On ADD/UPDATE:
1. Application requests data.	1. Application sends data.
2. Is currency on?	2. Is currency on?
3. If yes, run currency trigger.	3. If yes, run currency trigger.
4. Currency Trigger calls TER, The TER:	4. Currency Trigger calls TER. The TER:
• Executes the business function	• Executes the business function
• Performs the business logic	• Performs the business logic
• Scrubs data accordingly	• Scrubs data accordingly
5. Return data to database, and then to application	5. Update database.

When passing Math_Numeric currency fields into a business function, the currency values in the respective data structure must be populated. Math_Numeric work fields that contain currency values also need the proper currency information.

You can copy currency information to controls (work fields or others) in event rules by using the system function Copy Currency Info. You can call the currency triggers from within an application's event rules or from another business function.

Setting Up Currency Conversion

If your business uses more than one currency, you must designate the method of currency conversion to use.

► To set up currency conversion

From the Multi-Currency Setup menu (G1141), choose Set Multi-Currency Option.

1. On System Setup, click General Accounting Constants.
2. On General Accounting Constants, enter a value in the following field:
 - Multi-Currency Conversion (Y, N, Z)

The currency conversion flag is stored in Company Constant table (F0010) in the CRYR field for Company 00000.

You can set the Multi-Currency Conversion option to N so that currency conversion does not occur in JDB and the runtime engine.

Showing Currency-Sensitive Controls

When you design an application, you can decide whether to hide or show currency-sensitive controls, such as check boxes and radio buttons, at runtime.

► To show currency-sensitive controls

1. On Form Design, double-click the control that you want to appear on the form.
2. Click Control Options.

3. If you want to display currency fields, verify that the No Display if Currency is Off option is turned off.

When the No Display if Currency is Off option is on, currency-sensitive controls do not appear. If the No Display if Currency is Off option is turned off, currency fields are visible.

You must exit your current EnterpriseOne session and begin a new one to apply currency conversion changes.

Creating a Currency Conversion Trigger

If the table that you are using contains currency fields, you must specify how many decimal places exist in each column. When the source or destination fields are currency fields and you have not created a currency trigger, problems might arise if the value is used in a calculation. If you do not create a currency conversion trigger, the system cannot determine where to locate the decimal within a field.

► To create a currency conversion trigger

1. From Object Management Workbench, check out the table to which you want to attach event rules.
2. Ensure that the table is highlighted, and then click the Design button in the center column.
3. On Table Design, click the Design Tools tab, and then click Start Table Trigger Design Aid.
4. On Event Rules Design, choose the Currency Conversion event and attach the currency trigger that you want to use.
5. Click the Business Functions button.

The Business Function Search form appears.

Use the QBE line to search for selected business functions. You can use Category CUR or System Code 11 to find existing currency business functions. To read notes that describe the purpose of the business function, its parameters, and program requirements, click the Attachments button.

6. Choose the business function with which you want to work, and then click Select.
7. On Business Functions, attach the table columns to the business function structure, and then click OK.

The available objects that appear are for table column only.

8. On Event Rules Design, click Save, and then click OK.
9. On Table Design, click the Table Operations tab, and then click Generate Table.
10. Choose the data source for the table, and then click OK.
11. On Table Design, click the Design Tools tab, and then click Build Table Triggers.

The system creates the table event rule. The newly created or modified table event rule functions are now called from the database APIs whenever the corresponding event occurs against the table.

Tips of the Day

Many PeopleSoft applications provide tips of the day, which are sets of short informational text that appear each time the user launches an application or accesses a form. You can change these tip sets or create your own. Tips of the day appear sequentially, so the user can browse through the tips. When the user closes the tip form, the system records where in the tip sequence the user is and displays the next tip when the user launches the object again.

Working with Tips of the Day

In EnterpriseOne, tips of the day are the glossary texts of data dictionary items. You create one data dictionary item for each tip. Since you can translate data dictionary glossaries, tips of the day can appear in different languages.

You can associate tips with an application, a form, or an application version. The tips appear in the order that you specify, and you can override a user's option to turn off the tip of the day feature for the tip set.

After you have associated tips with an object, you can rearrange the tip order, add new tips, or delete existing ones from the tip set.

Prerequisite

- ❑ Create a data dictionary item with glossary text for each tip of the day. See *Creating a Data Item* in the *Development Tools Guide* for more information about creating data dictionary items.

► To work with the Tip of the Day utility

From the System Administration Tools menu (GH9011), choose Tip of the Day.

1. On Work With Tips of the Day, click Find to see the objects to which tips have been added.

To limit your search, enter an application name in the Application Name field.

A tree-style file structure appears on the left side of the form. Each object has its own folder. Tips can be associated with applications, forms, or application versions. Folder names include the application, form, and application version, in that order. For example, `RDA\` indicates the RDA application alone. `P0911\V0911I\ZJDE0001` indicates form V0911I in the ZJDE0001 version of application P0911.

2. To see the specific tips associated with an object, expand the folder for the object.

The data dictionary items associated with the object as tips appear in the file structure.

3. To view the text for a tip, click the tip.

The glossary text associated with the data dictionary item appears on the right side of the form.

4. To change the order of the tips or to add or delete tips for an object that already has tips associated with it, double-click one of the tips under the object.

The Tips of the Day Revisions form appears.

► To add tips of the day to an object

You can add tips to an application, form, or application version that does not already have tips associated with it. To add tips to an existing tip set, double-click one of the existing tips on the Work With Tips of the Day form to access the Tips of the Day Revisions form.

1. On Work With Tips of the Day, click Add.
2. On Tips of the Day Revisions, complete the following fields:
 - EnterpriseOne Tool
 - Force tip to all users
3. In the detail area, add a data dictionary item for each tip that you want to associate with the application. Complete the following columns for each row:
 - Tip Sequence
 - Data Item

Note

You cannot add a data dictionary item to the tip set if it does not have glossary text.

4. Click OK.

Messaging

Use EnterpriseOne messaging features to communicate pertinent information to the end user in the most effective and user-friendly way. When you design an application to use messaging, you must evaluate what information is necessary to enable a user to accomplish a task. You can deliver a message in real time, whereupon the message is displayed on the status bar. The method that you use to provide information to the user depends on the situation. For example, you can do the following:

- Use an interactive error message if the system encounters an error during the entry of a record.
- Use an informational message that the system sends to the Workflow Center if information needs to be conveyed and responded to.
- Use an alert message if information is urgent and requires immediate attention.
- Use a batch error message if the system detects errors in a batch process, such as while a report is running.

The three components of creating system-generated messages are as follows:

- The message itself
Do you require a simple message or a text substitution message? Are all of the text substitution pieces available?
- The logic that applies to the message
Has certain criteria, such as event rule logic, been met so that a message should be sent?
- The message type
Does the message require action by the users? Are all of the required parameters available at the time the message is to be sent?

Message Types

The system provides two main types of messages, as follows:

- Error and warning messages
- Information messages
Information messages can also be action messages that enable the user to connect from a current form to another form that will allow them to correct an error, or to evaluate information and then take action. To use action messaging, you must ensure that the parameters for the connecting form are available at runtime.

Within these message types you can use simple messages or text substitution messages. Text substitution messages allow you to use variable text substitution. The system inserts substitution values into the message for the appropriate variable at runtime. This provides the user a customized message unique to every instance of the message.

Text substitution messages are of two types, as follows:

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

Both types are created in the same manner.

Error Messages

Error messages are used to inform users that information was entered incorrectly (for example, a 3-digit UDC code that doesn't exist was entered) or if the system encounters technical difficulties retrieving information. Typically, error messages tell the user how to proceed.

Error messages are stored in the data dictionary. The primary data dictionary design tool is the Work With Glossary Items application (P92002).

Workflow Messages

In addition to sending and receiving internal and external messages, you can receive an active message, which is a type of message that a system workflow process automatically sends to a recipient.

Active Messages Workflow processes sometimes generate messages that require you to take action, such as approving or rejecting a change to a customer record. A lightning bolt icon identifies an active message.

Active messages contain a shortcut icon that links directly to an application. When you click the shortcut icon, the system retrieves the most current information from the database, which ensures that you get accurate information even if changes are made after an active message is sent to you.

You can set up a workflow process to send active messages to specific queues.

Level Messages

Level messages are used to categorize error messages into the proper level break within a report. They are like a container for messages. Level messages can be thought of in terms of "Here are your batch errors" or "Here are the document errors," and so on. Level messages are used to separate every logical grouping of error messages. Messages that begin with "LM" are level messages. Level messages that belong to glossary group "Y" indicate that they are workflow messages.

Information Messages

Messages that are not level messages ("LM") but are in glossary group "Y" are considered informational messages. These messages supply pertinent information to the user and usually require that action be taken.

Understanding Error Handling

Use error handling to prevent your program from being shut down by the operating system.

Event-Driven Model

Interactive messages appear whenever a specific event occurs. This means that errors clear, set, and display based on certain events. When an error occurs within an event, the event then runs again to clear the error.

For example, suppose the user types an incorrect value in a field. An error is set when the *Control is Exited* event is run (if you have assigned logic on this event or a high-level trigger is causing validation of the field). The user receives notification of the error through a visual cue (the field in error typically appears in red). In addition, the status bar displays information about the error. For more information about why the error occurred and how it can be corrected, the user can press the F8 function key (or the F7 function key if multiple errors occur and the user needs to view the errors on a form in sequence).

The user corrects the error by entering a valid value, and then exiting the field. This causes the same *Control is Exited* event to be run again, which clears the error and sets it again in the event that another incorrect or invalid value is entered again. If the value entered is valid, no error is set.

When is an Error Set?

When a user enters data in a form field or grid cell and presses the tab key to move out of the field, the runtime engine validates the value that was entered. You set an error so that if the validation process fails, (for example, if a value is not found in the appropriate UDC table, or an invalid date has been entered), the system issues an error with the appropriate error code and highlights the field. The system displays the error message and the message count on the status bar.

How Does the Application Know Which Field to Highlight?

When the system issues an error, the handle is on the field in error. If the error is within a grid, then the system identifies the row and column coordinates of the particular grid cell. The system stores this information in an `Error_Ctrl_Key` structure, along with the application form handle. When the system displays the error, it changes the color of the control and displays the error information from the error link list. If you use the system function *Set Control Error* or *Set Grid Cell Error*, you can specify the field or cell to highlight.

How an Event ID is Used for Error Setting

EnterpriseOne applications are event-driven. Each keystroke or mouse click is considered an event. When a user enters a value in a control and tabs out of the field, the *Leave Control* event or *Leave Column* event validates that field. All errors are set using the event ID in the `Error_Event_Key` structure. This event ID is used to clear the errors that were set on this event when the event is reprocessed. For example, when the user enters an invalid address book number and tabs out of the field, an error is displayed. This error is set with the Control Handle, Event ID, and Grid Cell Coordinate (if any). To clear this error, the user must enter a valid address book number and tab out again. The error is cleared if no more errors are found on this control.

Error Setting

Errors can either be set automatically or manually.

Automatic Error Setting

The runtime engine validates a field and automatically sets errors in the following instances:

- The item is a data dictionary item for which an edit rule trigger is defined in the data dictionary. In this case, the field for the data item is validated across all EnterpriseOne applications. Data dictionary triggers can also be assigned or redefined using overrides within the Form Design tool. However, in this case the trigger for the data item is specific to the application where the override was defined.
- Validation is performed when the Control is Exited event, Column is Exited event, or OK Button Processing occurs. You can use validation to check for invalid data or invalid dates.

Manual Error Setting

You can use either a system function or a business function API to set other error messages.

System Function

You can use a system function for setting an error message. The following system functions are available for this purpose:

- Set Edit Control Error
- Set Grid Cell Error

You use *Set Edit Control Error* to set an error on a Form Control field through event rules. You pass in the following parameters as system function arguments:

- Control (for example, the field that is in error)
- Error Code (a literal in this case means the error message created through the Data Dictionary Application (P7992002) or a variable)

You use *Set Grid Cell Error* to set an error on a Grid Control field through event rules. You pass in the following parameters as system function arguments:

- Grid (does not receive any value)
- Row Number
- Column Number
- Error Code

System functions cannot use data items with text substitution.

Business Function API

You use business functions to manually set error messages. Use one of the following business function APIs to set an error message **without** text substitution:

- JDERTN(void) jdeErrorSet(LPBHVRCOM lpBhvrCom, LPVOID lpErrInfo, ID idItem, PJSTR lpzError, LPVOID lpDs)

The last parameter is always NULL.

- jdeSetGBRError(lpBhvrCom, lpVoid, (ID)0, _J("Error#"))

The following example is an error message without text substitution:

```
{
    jdeSetGBRError(lpBhvrCom, lpVoid, (ID)0, _J("4363"));
    return ER_ERROR;
}
```

The return code successful message has no influence on what happens next. It is informational only.

Use one of the following business function APIs to set an error message **with** text substitution:

- jdeErrorSet(lpBhvrCom, lpvoid, idItem, lpzError, lpDs)

This sets errors through business functions using substitution text. It always gets called.

- JdeSetGBRErrorSubText(lpBhvrCom, lpvoid, idItem, lpzError, lpDs)

JdeSetGBRError and jdeSetGBRErrorSubText work the same except that jdeSetGBRErrorSubText has an additional parameter in which you pass the address of the data structure that holds the information you want substituted in.

The following example is the setup for a text substitution error message using a data structure:

```
/* *****
* Declare structures
***** */
DSDE0022    dsDE0022;

/* *****
* Main Processing
***** */
if(idJDBReturn != JDEDB_PASSED)
{
    jdeStrncpy(dsDE0022.szDescription, (const JCHAR *) (_J("F41001")),
               DIM(dsDE0022.szDescription));
    jdeSetGBRErrorSubText(lpBhvrCom, lpVoid, (ID)0, _J("078D"), &dsDE0022);
    JDB_FreeBlvr(hUser);
    return ER_ERROR;
}
```

In this example, DSDE22 is the data structure for the error messages. It is defined in jdeapper.h, which is where all of the data structures are defined for text-substituted error messages used by EnterpriseOne. Non-EnterpriseOne-substituted error messages should be defined in their own global declaration header file or in the function header file that is using the structure.

A business function can call a validation routine (jdeddValidation), which when called might set an error if the field in question is invalid.

To use text substitution error messages in C business functions, you create an instance of a data structure in the global header file for every custom business function. If the error data structure exists, it should exist in jdeapper.h.

Resetting Errors

The system can reset errors on the OK button or on the Find button. This action occurs on both manual and automatic errors.

Resetting Errors on the OK Button

The system resets errors when the OK button is clicked on a header detail form or a headerless detail form. The following actions occur:

1. Setup Error_Event_Key structure with event *Button Clicked*, the control handle is the OK button.
2. Clear event errors on OK *Button Clicked*.
3. Clear error due to the following events:
 - *Button Clicked Processing Done*
 - *Add Record to DB - Before*
 - *Add Record to DB - After*
 - *Update Record to DB - Before*
 - *Update Record to DB - After*
 - *All Grid Recs Added to DB*
 - *All Grid Recs Updated to DB*
 - *Delete Record from DB - Before*
 - *Delete Record from DB - After*
 - *All Grid Recs Deleted from DB*
 - *Delete Grid Rec Verify - Before*
 - *Delete Grid Rec Verify - After*
 - *Row is Exited*
4. Stop processing if error still exists.
5. Perform event rule on *Button Clicked*.
6. ValidateAllData - including form controls.
7. Stop processing if error occurred.

8. Delete any rows that were removed from the grid.
9. Perform event rule for Add or Update to Database Before or After.
10. Perform grid changes (validate any unprocessed grid row).
11. Stop processing if error occurred.
12. Perform event rule on *After Button Clicked*.
13. Stop processing if error occurred.
14. Otherwise, clear screen and exit.

Note

Do not try to validate and set errors on any grid control field for the *Button Clicked* event on the OK button. The *Button Clicked* event will not process all the grid rows.

Resetting Errors on the Find Button

The system resets errors on the Find button. It performs the following steps:

15. Setup Error_Event_Key structure with event *Button Clicked*.
16. Clear event errors on OK/Select *Button Clicked*.
17. Clear errors due to the following events:
 - *Button Clicked*
 - *Last grid record has been read.*
 - *Form Record is Fetched*
 - *Grid Record is Fetched*
 - *Confirm Delete - Before*
 - *Confirm Delete - After*
18. Perform event rule on *Button Click*.

Multilevel Error Messaging for C Business Functions

When a business function calls another business function and the error is issued from the second business function, you must provide a mapping key array. Otherwise, errors will be set incorrectly.

When you use `jdeCallObject` (the standard API for calling other business functions from within a business function) for the second business function call, the sixth parameter is for error mapping. Each business function has its own header and definition. You need to know why you are calling the second level business function. For example, assume that you are calling a second level business function to validate the company number. You must have the company number ID in the first business function header so you can determine the company number ID in the second business function.

► **To create multi-level error messaging**

1. Open the header file for the called business function and determine the maximum number of possible mapping fields.

Calling Function's Header File	Called Function's Header File
#define IDERRszComputerid_1 1L	#define IDERRcHeader_1 1L
#define IDERRszCompany_2 2L	#define IDERRcEvent_2 2L
#define IDERRszDate_3 3L	#define IDERRDetail_3 3L
#define IDERRszValue_4 4L	#define IDERRCompany_4 4L
#define IDEERszPoint_5 5L	#define IDERRDateOpen_5 5L

2. Create an Error map section in the C file for the calling business function.

In the following example of an active error message, you need to map one field (IDERRCompany_4).

```

/*****
* Business Function: B1234
*
*
* Parameters:
*      LPBHVRCOM
*      LPVOID
*      LPDS1234
*****/
/*****
* Error Mapping Section
*
* Map to function "ValidateCompanyName"
* #define ND0400290 2
*****/
#define ND0400290 2

```

- ND 0400290 is the data structure number of the called business function. The number 2 is the number of mapped fields.

- Before each #define statement, include a comment that refers to the business function name for which this number will be used.
- In the variable section, create a *cm_xxx* map array for each function that needs to return errors.

For example:

```

/*****
* Variable declarations
*****/

CALLMAP      *      pcmD0400290      = (CALLMAP *)NULL;
CALLMAP      cmD0400290_1 [ND0400290]
= { { IDERRszPaymentTermsCode_3, IDERRszPaymentTermsCode_2 },
    { IDERRjdDateInvoiceJ_2, IDERRjdTransactionDate_9 } };

```

- The array is mapped from IDERRszPaymentTermsCode_3 to IDERRszPaymentTermsCode_2.
- The function call to jdeCallObject is:

```

IdReturnCode = jdeCallObject(_J("ComputeDueDate"),
                             NULL, lpBhvrCom, lpVoid, (void *)&dsD0400290,
                             pcmD0400290, (int) (ND0400290), (JCHAR *)NULL,
                             (JCHAR *)NULL, (int) 0);

```

The business function sets the error on the ID field.

Working with Error Messages

EnterpriseOne automatically displays messages based on certain events. For example, you can display an error message whenever an invalid value is entered into a field.

EnterpriseOne uses the data dictionary glossary to display messages. The message is contained within the glossary portion of a data item. By using the existing framework of the data dictionary, you do not have to create a separate messaging system.

Locating an Existing Error Message

Whenever possible, you should use an existing message instead of creating a new one. Use Work with Glossary Items to locate an existing message. You can narrow your search by using glossary group types and descriptions similar to the error message that you want.

► To locate an existing error message

From the Data Dictionary Design menu (GH951), choose Error Messages.

1. On Work With Glossary Items, complete any of the fields in the header area to filter your search to specific error messages, and click Find.

2. Choose the message that you want to use, and click Select.
3. Click the Item Glossary tab to preview the text for the error message.

Creating a Simple Error Message

Simple error messages (static messages) contain literal text-- for example, "Enter a valid date." Simple error messages do not use text substitution.

Adding an Interactive Message Data Item

You can create a new data item or choose an existing item to display a message. Before the error message is functional, you must connect it to a form control by using event rule logic.

```
jdecIndex[1].CacheKey[4].nOffset=offsetof(I1000042,szSubledger);  
jdecIndex[1].CacheKey[4].nSize=DIM(dsI1000042.szSubledger);  
jdecIndex[1].CacheKey[4].idDataType=EVDT_STRING;
```

► To add an interactive message data item

From the Data Dictionary Design Menu (GH951), select Error Messages.

1. On Work With Glossary Items, click Add.
2. On Glossary Items, click the Item Specifications tab, and then complete the following fields:
 - Glossary Group
 - Product Code
 - Description
 - Error Level
3. Click OK.

Creating a Text Substitution Error Message

A text substitution error message is an error message that allows variable text to be substituted directly into the glossary text for the error message. This allows you to display the same basic message with certain variable values.

For example, assume that a user enters an invalid search type in a field. The error message "Search type xx is not contained in the 00 ST table" appears. The system will replace the xx variable in the message by the search type that the user entered.

If you want to use text substitution error messages in batch processes, you must create a business function to pass values to the data dictionary data structure. The event rule engine (set user error) will not pass values to the text-substituted error message.

► To add an interactive message data item using text substitution

From the Data Dictionary Design menu (GH951), select Error Messages.

1. On Work With Data Items, choose Glossary Data Item from the Form menu.
2. On Work with Glossary Items, click Add.
3. On Glossary Items, click the Item Specifications tab, and then complete the following fields:
 - Glossary Group
 - Product Code
 - Description
 - Error Level

When you type the description, use & followed by a number for each subset variable.

Now you are ready to define the glossary text to display at runtime.

Defining the Glossary Text for an Interactive Message

After you add an interactive message data item, you must add a glossary for that item.

► To define the glossary text for an interactive message

From the Data Dictionary Design menu (GH951), select Error Messages.

1. On Work With Glossary Items, choose Glossary from the Row menu.
2. Type the text of your message.

Use &x to assign variables in the text. These variables correspond to members in a data structure.

The order of the data structure is the index of the &x structure. If the structure contains four members, assign &1 to the first member, &2 to the second, and so on. For example:

Description = &1

UDC Value = &2

System = &3

Type = &4

3. When your text is complete, click OK.

Attaching a Data Structure Template to a Message

Each text substitution error message requires a corresponding data structure. If the data structure does not already exist, then you must create it. You must ensure that the members in your message match the members in your data structure.

See Also

- ❑ See *Data Structures* in the *Development Tools Guide* for information about adding a data structure

► To attach a data structure template to the message

1. On Glossary Items, click the Data Structure Template tab.
2. To locate an existing structure, click Text Substitution, and then the Browse button.
3. On Individual Object Search & Select, choose a data structure and click OK.

If no existing structure meets your needs for the interactive message, you must create one.

Attaching an Interactive Message Data Item

To make an interactive error message appear at runtime, you must attach a system function. If you are using text substitution, you cannot attach a message data item.

► To attach a message data item

1. On Event Rules Design, click the system function button.
2. Choose the system function that you want to use.
3. Use available objects to complete the control value in Parameters.
4. Choose <Literal> for the Error Code Value in Parameters.

Use the data item name of the data dictionary item that contains the error message that you want to appear for the literal value.

Working with the Send Message System Function

The Send Message API is associated with the following tables:

- F01131 - PPAT Message Control File
- F01131M – JDEM Multi Level Message
- F01133 - PPAT Message Detail File
- F00165 - Media Objects Storage

The PPAT Message Control File table and the PPAT Message Detail table (F01133) are used by the Message Center application for delivery of messages. They contain information about who is to receive the message, the current status of the message, who sent the message, what date the message was sent, which mailbox the message belongs to, and all other pertinent mail delivery information. The Media Object Storage table (F00165) contains the subject and body of the message.

The Send Message system function is made up of three components:

- The addressing and destination
- The message, with or without text substitution
- The action message, the act of calling other programs from a message

The following table lists the parameters and their functions for the Send Message system function.

Parameter	Description	Typical Values	Mandatory
Address Number	The address book number of the person who is to receive the message.	Address Book Number - Data Dictionary item AN8	Yes
Mailbox	The mailbox to which the message should be delivered (for example, "Personal in Basket")	The two-character field associated with a Mailbox, which is found in UDC 02/MB.	Yes
*Subject	The subject of the message.	A brief description of the message, or Blank if a message template is used.	No Choose "None" if a message template is to be used.
*Text	The main body of the message.	The message that will be sent, or Blank if a message template will be used.	No Choose "None" if a message template is to be used.
Active	A value that indicates whether a connection exists between the message being sent and another form.	When this item is used, the user is prompted through the form interconnect process.	No
DDMessage	The message template from the data dictionary.	Any valid data dictionary item whose glossary can be used for a message.	No
MessageKey	A returned parameter that holds the key value for the message sent.	Data dictionary item (for example SERK).	No

If the Subject and Text fields are used in conjunction with the DDMessage field, then the DDMessage (MessageTemplate) will be concatenated with the Subject and Text fields.

► To use the send message system function

1. On Object Management Workbench, check out the table with which you want to work.
2. Ensure that the table is highlighted, and then click the Design button in the center column.
3. On Table Design, click the Design Tools tab, and then click Start Table Trigger Design Aid.
4. Click System Functions, and then click the Function Selection tab.
5. In the Function Selection pane, choose Messaging, and then choose Send Message.
6. Under Parameters, complete the Recipient field to designate who is to receive the message.

This field typically uses an address book number. You can also use an e-mail address here.

7. Designate the Mailbox.

The mailbox indicates what type of message is being sent. Mailboxes are stored in the UDC table 02/MB. Mailboxes from UDC table 02/MB appear when focus is on the Mailbox field.

8. Click the data item called Active, and then choose Define Active Message to begin the form interconnection process.

The active message portion of the message works similarly to form interconnections from event rules.

Defining an Active Message

Complete the following task to define an active message.

► To define an active message

1. Double-click Define Active Message.

Work with Applications appears.

2. Choose the application with which you want to work.

3. Choose the form with which you want to work.

4. Choose the version with which you want to work (if necessary).

The Processing Options form might appear if processing options are attached to the form.

The Form Interconnections form appears, on which you can assign all of the values to be passed.

5. Designate the values that you want to pass and then click OK.

6. On System Functions, choose Define Message.

The system prompts you to choose a message.

7. Enter the appropriate alias in the Dictionary Item field and click Find.

The substitution points are numbered &1, &2, and so on. The parameter numbers correspond to the substitution numbers. Substitution variables can appear in both the description and the glossary of the message.

8. Map the appropriate objects to the substitution parameters to complete the message.

Batch Error Messages

The error message system gives users a consistent interface to review errors when working with batch programs. When a batch program has finished processing all messages regarding the success or failure of a job, the system sends a message to the user. To enhance the usability of the messages, the system uses a tree structure (or parent/child structure) to group related messages. To provide additional flexibility and functionality, you can use text substitution, and you can make a message "active," meaning that the user can open an associated form by clicking within the message.

Batch Error Messaging

The Work Center displays the error messages that appear after a batch job has completed. When you create these batch error messages, you need to determine the possible messages EnterpriseOne users will need. For example, you might create a number of different messages that are generated when a journal entry report is run. You can create a message stating that the report completed normally if the report balances. Additionally, you can create multiple levels of messages describing various errors if the report is out of balance. The first level might state that the report completed with errors, and additional levels would explain the specific details about the errors.

Level-Break Messages

A level-break message is a message that acts as a container for other error messages that are produced by the batch process. Level-break messages can be action messages, which contain a shortcut to an application and require action on the part of the user. They can also be messages that require no action, but that might instruct the user to review some information.

To understand the role of level-break messages, you must examine how errors occur in batch jobs. Most errors are created by edits that occur at level breaks in the entire batch process. You typically want to group any errors that might have occurred at these different level breaks. The mechanism that you use to group these errors is the level-break message. This implies that for each reporting program that uses the Work Center application program interface (API) to manage errors, you need to create one level-break message for each phase of level-break.

Understanding How Level-Break Messages Work

Level break and action messages are used to group (or package) one or more errors.

All individual messages are level-break messages except for the final error messages. The Work Center API creates and manages Job Completed level-break messages, but you create all other messages. The level-break messages are not error messages. They are packaging or categorizing messages that you set up to communicate to the user information about in the batch, document, line, and so on, in which the error occurred. You set these messages using the `jdeSetGBRError` or the `jdeSetGBRErrorSubText` function.

For example, "Job R89004 ZJDE0001.c" and "Batch 3230 G in Error" are level break messages. "Job R89004 ZJDE0001.c" is a level-one message that is automatically generated by the system and "Batch 3230 G in Error" is a level-two message. "AA 20050606 3031" and "Intercompany Out of Balance" are actual error messages.

The level-break message consists of two distinct components. The first component of a level-break message is the text for the message. The second component indicates whether the message will be an action message.

All level-break messages contain the text component, but they might not all be enabled as action messages. The decision to enable the action message piece of the level-break message is left to the report designer.

You use appropriate level breaks within your design to group related messages. The Work Center API should be called at these level breaks. A level break usually should be set up any time that a record is no longer processed and a new record is about to be read. This concept is consistent even if the report has several nested record breaks.

The following topics show the how messages might appear when an out-of-balance journal entry upload has completed.

First-Level Messages

First-level messages appear when users open their personal in-baskets. A plus symbol next to the message indicates that additional levels of messages exist beneath it. First-level messages might show the name of the batch job, explain that it completed with errors, and instruct the user to review the details about the errors.

Second-Level Messages

Second-level messages appear when the user double-clicks the plus sign next to the first-level message. In this example, the second-level message informs the user that a specific batch number needs to be reviewed.

Third-Level Messages

Third-level messages appear when the user double-clicks the plus sign next to the second-level message. In this example, the third-level message informs the user that the batch job completed with errors because it was out of balance, and then provides several solutions to resolve the problem.

See Also

- ❑ *Working with Level-Break Header and Footer Sections* in the *EnterpriseOne Report Writing Guide* for more information about level breaks within sections

Text Substitution Error Messages

Error messages must be as informative to the user as possible. You can accomplish this through text substitution, which allows the system to embed variable text (such as dates, amounts, and so on) into a message at run time. You set up text substitution messages using the data dictionary, which helps to ensure consistency of jargon and terminology. For example, the message "Voucher Batch 2453 contains errors," uses the value 2453 as a parameter to the message. This value is substituted at runtime, and the rest of the information from the message is stored in the data dictionary error glossary. This message is only the short description in the data dictionary. When you open the message, the complete data dictionary glossary with text substitution appears.

See Also

- ❑ *Creating a Text Substitution Error Message* in the *Development Tools Guide* for more information about creating interactive and batch message data items, defining the glossary text for a run time message, and creating a new data structure for the message

Action Messages

After users review errors and determine resolutions, they typically need to access a revision program to resolve the errors. You can set up specific level-break messages known as action messages that provide users the ability to resolve the errors by calling the corresponding revision program directly from the Work Center. Action messages call an EnterpriseOne application and pass the required variables to that application. For example, the user would be able to automatically load the Voucher Revision form with the record in error by clicking a shortcut within the error message. You determine the appropriate application and the correct values that need to be passed to that application. The system highlights action messages in the grid to differentiate them from nonaction messages.

Work Center API

When you call the Work Center API, it assumes a child/parent order. In other words, when the API is called, it assumes that any error that is in the runtime error message stack belongs to the level sent into that instance of that API call. This means that all of the errors in the error space at that time, whether they have been set through business functions or through EnterpriseOne event rules, are packaged or grouped together as children of the level that was passed to the Work Center API. These error messages are then cleared from the error space so that the next group of messages can be made based on a new set of records.

The timing of the calls to the Work Center API is critical. The reporting program likely will start by editing the header-level record, which will lead to a set of detail records. The detail records are the first to be read and processed. Thus, the calls to the Work Center API will likely send level-break numbers in descending level-break order.

For example, the actual series of level-break calls to the API might look as follows:
4,4,4,3,4,4,3,2,4,4,3,2,1.

This series indicates that the call structure started four levels down. The first call at level 4 let the Work Center API find any messages that had occurred at that time and create child messages using the level 4 message as the parent. If no errors had occurred, then no messages would have been created. This call sequence example shows that the API was called at level 3 after three calls to level 4. When the call to level 3 was made, the Work Center API remembered if any level 4 messages were written or not. In other words, if no errors occurred when any of the level 4 calls were made, then the Work Center API would not create the level 3 messages. If at least one error existed at any of the level 4 calls, the level 3 and the level 2 message would be created.

You must call the Work Center API at every level. Because the Work Center error messages are created based on a parent/child structure, if a level call is skipped, then the API has no way to group the child messages and child levels that are already created.

For example, the following level call structure is valid: 6, 6, 5, 4, 3, 4, 4, 3, 2, 1. The call sequence 6, 6, 4, 3, 4, 4, 3, 2, 1 is invalid because when level 6 is called, no corresponding call to level 5 exists.

The Work Center API must be called with a level 1 when the reporting job is about to complete. Hence, level 1 is the parent to all errors and level-break messages. It issues the "job completed" message. The level 1 call to the Work Center API is essential. A level 1 call to the API ensures not only that no orphan Work Center records are created, but also cleans up all allocated storage used by the Work Center system. The level 1 call to the API should occur only once in the report, typically in the End Section event of the primary section of the report.

Creating a Level-Break Message

Level-break messages act as a container for error messages. You can create level-break messages that are action messages or nonaction messages.

You create level-break messages by creating data dictionary items, error data structures, business function error structures, and business functions.

See Also

- ❑ *Action Messages* in the *Development Tools Guide* for more information about active and nonactive messages

Creating a Data Dictionary Item for a Level-Break Message

The data dictionary item is the text portion of the level-break message, as it appears in the work center. Before you create a data dictionary item, you should review the level-break messages that have already been created to determine whether a level-break message that you want to use already exists.

► To create a data dictionary item for a level-break message

1. Using Work With Data Dictionary Items, search for business functions in the list of existing level-break messages.

For every level-break message created, a data dictionary item and at least one business function are also created to reference that item. To determine the level-break messages that already exist, use query by example rows in the Object Management Workbench (OMW) to locate all business functions (object type BSFN) that start with the letters BLM.

The BLM naming convention refers specifically to business functions for level-break messages created by EnterpriseOne. PeopleSoft recommends that you use the following criteria to name your own business functions for level-break messages:

BLMxxyy

Where:

xx = The system code that you are using (between 50 and 55).

yy = A unique number. This unique number should be carried forward to other items. The maximum number of characters is 8.

Note

Use the Description field to help you decide which existing messages might work in your report.

If you find several business functions for level-break messages that are appropriate for your report design, write them down and verify the text within the data dictionary item itself. Use the data dictionary to inquire on data items that begin with LM and are followed by the same numbers that followed the corresponding BLM function name.

The LM naming convention refers specifically to level-break messages with text substitutions that are created by EnterpriseOne. These are classified under glossary group Y. PeopleSoft recommends that you name your own level-break message with text substitutions using the following criteria:

Lxxyy

Where:

xx = The system code that you are using (between 50 and 55).

yy = A unique number. The maximum number of characters is 8.

For example, if business function Set Level A/R Receipt Pre-Post error (BLM0025) is a level-break message for which you want to see the description, inquire on data item Receipt &1 Customer &2 In Error (LM0025).

2. If the text description meets your criteria, verify whether any substituted variables that the level-break message uses are valid for your circumstance.

If the text in the LM message contains substituted variables (such as &1), you then need to verify the data items of the data structure used by the level-break message. The data items in the structure must be the exact data items that you will be passing in your report.

To verify that the data item is in the data structure, use OMW and inquire on data structure DELM followed by the same numbers that you used before.

The DELM naming convention refers specifically to level-break message data structures created by EnterpriseOne. PeopleSoft recommends that you name your own level-break messages using the following criteria:

DELMxxyy

Where:

xx = The system code that you are using.

yy = A unique number. This unique number should be the same number that you used previously. The maximum number of characters is 8.

For example, DELM0025 would be the data structure created for the text substitution of level-break message LM0025. You might need to check out the structure to view individual fields.

Creating a Data Structure for the Data Dictionary Item

Each level-break message created requires a corresponding data structure to go with it. The data structure must be defined with the exact data items that will be substituted when the message is displayed. For example, a level-break message describing a particular error for Address Book Number would imply that the corresponding data structure would have data item AN8, the unique number that identifies an entry in the Address Book system. This restriction implies that many new data structures are going to be created.

To create the data structure, use OMW. The name should use the same unique number that was used when creating the Lxx data dictionary item. This number is to be appended to DELxx. For example, level-break message L55025 would use DEL55025 as the name of the data structure.

You will also need to create a typedef for this data structure. You include this typedef in the business function that you create. The typedef converts the data structure to C so it can be used in the business function.

See Also

- *Data Structures* in the *Development Tools Guide* for more information about data structures

Creating a Data Structure for a Level-Break Message Business Function

You create business functions to deliver level-break messages. Each business function that you create needs to have several standard parameters in addition to the variables that are used for the text substitution and the variables needed for the message to be active. This means that you must create a second data structure to pass all of these variables.

The DLM naming convention refers specifically to PeopleSoft-created business function data structures. PeopleSoft recommends naming your own level-break messages using the following criteria:

Dlxxyy

Where:

xx = the system code that you are using (between 55 and 59).

yy = A unique number. This unique number should be the same number used previously. The maximum number of characters is 8.

For example, the EnterpriseOne-created level-break message LM0025 would use DLM0025 as the name of the business function data structure. You must have a business function for each level-break message. Do not confuse this data structure with the data structure that was created for the data dictionary item in the previous procedure. The difference between the two is that the data structure for the business function is used to move data variables to the level-break function. The data structure used for the data dictionary item is used to store data that is mapped to the glossary for a particular data dictionary item.

When you create the data structure for the business function, include the following items:

- All data items used for the level-break text substitution message.
- All data items needed for the message to be active (that is, any variable or variables needed to load the form data structure for a given application). Any data items that are used for the form interconnection only need to be renamed so that the letters FI_ appear after the Hungarian prefix; for example, jdFI_GlDate or mnFI_Openamount. If you are not making the message active, you do not need to include these items.
- Add data item ev01 and change the variable name from EventPoint01 to cIncludeInterconnect. This parameter is used as a flag to determine if the message is active. This parameter should be a common parameter to all level-break messages, even if the

original intention of the level-break message is not to call an application. This parameter allows you to use level-break messages in different applications, but not to launch applications. You must have a 1 in the data structure value to launch an application.

- Add data item *genlng* and change the variable name from *GenericLong* to *idGenlong*. You use this parameter to control all Work Center messaging. It is not intended to be used for anything other than a work field for the system.

Creating a Level-Break Business Function

After you have created the DLxx data structure, you create the business function that processes the level-break errors and performs all of the mapping for the active message.

► To create a level-break business function

1. Create an object in the Object Management Workbench for the new business function.

The name of the business function is the unique number preceded by BLxx. For example, if you named the data dictionary item for the level-break message Receipt &1 Customer &2 In Error (L55025), you would name the business function Set Level A/R Receipt Pre-Post error (BL55025). The EnterpriseOne naming convention is BLMxx.

2. Enter the name of the actual function on the Business Function Design form within Object Management Workbench.

When you name the function, the standard specifies that you start the function with the name `SetLevel_xx`, where *xx* is the system code. Append the function with other descriptive words to identify the purpose of the level-break message. For example, you might name the business function BLM0025.

3. In Business Function Design, attach the business function data structure to the function by highlighting the row and then choosing Parameters from the Row menu.

This data structure should be the DLxx data structure that you created earlier.

4. Choose Form Create to generate an .h file.
5. Copy and paste the `typedef` for the business function into its header file.
6. Create a `typedef` for the text substitution data structure DExx.

To create the `typedef` you will need to create an inactive function line.

7. After you create `typedef`, paste the data structure template into the Structure Definitions section of the business function header file.
8. Modify the source file for the level-break message by copying the source code of an existing BLMxxx.C business function.
9. After you copy the source into the new level-break business function, review the copied source and make appropriate changes.

Sample Source Code Highlights

The following sample of the shell source code shows which pieces need to be included and where you will be required to enter your own code for a business function.

You need to manually map fields from the business function's data structure to the `dsTextData` data structure; this is the data structure for text substitution in the level-break message. You also need to manually map fields from the business function's data structure to the `dsFormData` data structure; this is the data structure that is used for the active message.

Variable Declarations

In the "Variable declarations" section, the following lines should declare the level-break message variables:

```
/* *****  
 * Variable declarations  
 * ***** */  
JCHAR      szForm[11]; /* Name of form in application */  
JCAHR szDDitem[11]; /* Data dictionary name of the level message */  
JCHAR      szDLLName[11]; /* Name of the application DLL */  
JCHAR      szDsTpl[11]; /* Name of the text substitution data structure */
```

Declare Structures

In the "Declare structures" section, enter your own code for the appropriate type of level-break message. The following examples are from an existing business function:

```
/* *****  
 * Declare structures  
 * ***** */  
DSDELM0002 dsTextData; /* Instance of text substitution structure */  
FORMDSW0411Z1D dsFormData; /* Instance of form interconnect structure */
```

Set Pointers

In the "Set pointers" section, the following lines ensure that the level-break message will work:

```
/* *****  
 * Set pointers  
 * ***** */  
if (lpDS->idGenLong == (ID) 0)  
{  
    jdeSetGBRError (lpBhvrCom, lpVoid, (ID) 0, _J("4363"));  
    if (hUser)  
    {
```

```

        JDB_FreeBhvr(hUser);
    }
    return ER_ERROR;
}
else
    lpDSwork = (LPDS_B0100011A) jdeRetrieveDataPtr(hUser,lpDS->idGenLong);

```

Main Processing

The following lines should exist In the "Main Processing" section.

```

jdeStrcpy((JCHAR*)szDsTpl,(const JCHAR*)(_J("DELM0002")),DIM(szDsTpl)-1);
jdeStrcpy(szDDitem,(const JCHAR*)(_J("LM0002")),DIM(szDDitem));
memset((void*)(&dsTextData),(int)(_J('\0')),sizeof(dsTextData));

```

Assign Values from lpDS Data Structure to dsTextData Here

In the "Assign values from lpDS data structure to dsTextData here" section, enter your own code for the appropriate level-break message. When you assign values, you map the business function data structure items to the data dictionary data structure items. The following examples are from an existing business function:

```

jdeStrncpy(dsTextData.szEdiuserid,(const JCHAR*)(lpDs->szEdiuserid),
    DIM(dsTextData.szEdiuserid));
jdeStrncpy(dsTextData.szEdibatchnumber,(const JCHAR*)(
    (lpDS->szEdibatchnumber),
    DIM(dsTextData.szEdibatchnumber));
jdeStrncpy(dsTextData.szEditransactnumber,(const JCHAR*)(
    (lpDS->szEditransactnumber),
    DIM(dsTextData.szEditransactnumber));

```

In this example, dsTextData.szEditransactnumber is the data dictionary data structure item and pDS->szEditransactnumber is the business function data structure item. Use the Strncpy API for strings and the Mathcopy API for math numerics. If you use memcpy for dates, the characters are assigned directly.

Assign Values from lpDS Data Structure to dsTextData Here

In the "Assign values from lpDS data structure to dsTextData here" section, the following lines ensure that the level-break message will work:

```

if (lpDSwork->lpBlob->lpTSDSMPL != (LPDSTMPL) NULL)
{
    lpDSwork->
        lpBlob->lpTSTEXT=(PJSTR) AllocBuildStrFromDstmplName((LPDSTMPL)
        lpDSwork->lpBlob->lpTSDSMPL,(JCHAR*)szDsTpl,

```

```

(LPVOID) &dsTextData);

jdeStrncpy (lpDSwork->lpBlob->szDDItem, (const JCHAR *) (szDDItem),
            DIM(lpDSwork->lpBlob->szDDItem));
}

if (lpDS->cIncludeInterconnect == _J('1'))

```

Form Interconnect Processing

In the "Form interconnect processing" section, enter your own code for the appropriate level-break message. The following example is from an existing business function. In the "Form interconnect processing" section, the following lines ensure that the level-break message will work:

```

jdeStrncpy (szDLLName, (const JCHAR *) (_J("P0411Z1")), DIM(szDLLName));
memset((void *) (&dsFormData), (int) (_J('\0')), sizeof(dsFormData));
memset((void *) (szForm), (int) (_J('\0')), sizeof(szForm));
jdeStrncpy ((JCHAR *) szForm, (const JCHAR *) (_J("W0411Z1D")), DIM(szForm) - 1);

```

Assign Values from LpDS Data Structure to dsFormData

In the "Assign values from LpDS data structure to dsFormData" section, enter your own code for the appropriate level-break message. The following example is from an existing business function. This example shows how you pass information from the data structure for the business function to the data structure for the form:

```

jdeStrncpy (dsFormData.EDUS, (const JCHAR *) (lpDS->szEduuserid),
            DIM(dsFormData.EDUS));
jdeStrncpy (dsFormData.EDBT, (const JCHAR *) (lpDS->szEdibatchnumber),
            DIM(dsFormData.EDBT));
jdeStrncpy (dsFormData.EDTN, (const JCHAR *) (lpDS->szEditransactnumber),
            DIM(dsFormData.EDTN));
ParseNumericString (&dsFormData.EDLN, _J("1.0"));
dsFormData.EV01 = _J('1');

```

Get the Form Data Structure ID from the SVRDTL Table

In the "Get the form data structure id from the SVRDTL table" section, the following lines ensure that the level-break message will work:

```

If (JDESPECRESULT_PASSED==jdeSpecOpenLocalIndexed (&hTam, hUser,
                                                    JDESPECTYPE_SVRDTL, SPECKEY2_SVRDTL))
{
    jdeStrncpy ((JCHAR *) lpDSwork->lpBlob->szForm, (const JCHAR *) (szForm),
                DIM (lpDSwork->lpBlob->szForm) - 1);
    JdeStrncpy (Key.szForm, szForm, DIM(Key.szForm));
}

```

```

ASVRDtlData.DataType=JDESPECDATA_RAWBLOB;

JdeSpecFetchSingle(hTam,&ASVRDtlData,&Key,1);

If(ASVRDtlData.pSpecData !=(void *)NULL)
{
    lpASVRdtl=ASVRDtlData.pSpecData;

    JDBRS_GetSTMPLSpecs(hUser,(JCHAR*)lpASVRdtl->szFITemplateName,
                        &lpDSwork->lpBlob->lpFIDSMPL);

    if(lpDSwork->lpBlob->lpFIDSMPL !=(LPDSTMPL)NULL)
    {
        lpDSwork->lpBlob->lpFITEXT=(PJSTR)AllocBuildStrFromDstmplName(LPDSTMPL)

        lpDSwork->lpBlob->lpFIDSMPL,(JCHAR*)lpASVRdtl->szFITemplateName,
                                (LPVOID)&dsFormData);

        jdeStrncpy(lpDSwork->lpBlob->szDLLName,(const JCHAR *) (szDLLName),
                                DIM(lpDSwork->lpBlob->szDLLName));

    }

    jdeSpecFreeData(&ASVRDtlData);
}

jdeSpecClose(hTam);
}

```

Function Clean Up

In the "Function Clean Up" section, the following lines ensure that the level-break message will work:

```

if(hUser)
{
    JDB_FreeBhvr(hUser);
}

return (ER_SUCCESS);
}

```


Calling the Work Center Initialization API

After you create the business function, you must compile and check it in to the object librarian. Then, you must attach the Work Center APIs. The first step in attaching the APIs is to call an event rule in the Work Center. You usually call this on the init section in the parent section.

► To call the Work Center initialization API

1. Within Report Design, create a global work field. When creating the work field, use the data dictionary item GENLNG.
2. Call the Work Center initialization function. The report finds this business function. This function is named InitializePPATapi. (The actual name of the source file is B0100025.C.). F01131 is EditJDEM Error Message.

Generally, this function is called in the primary section of the report using the Initialize Section API.

Use the following table to identify which parameters to send:

Parameter	Description	Allowed Values	Typical Values
szUserid	User ID override. If not passed, the user who ran the UBE receives all messages. If this field is completed, it supersedes the following Address Number override parameter.	Leave blank or use any valid user ID. If left blank, all messages go to the submitter of the UBE.	Blank value
mnAddressnumber	User ID override. Use this field if you know the user address, but not the user ID for a user.	Leave blank or use any address number of a valid user. If left blank, all messages go to the UBE submitter.	Blank value
cDoNotLogWarnings	If this feature is turned on, no warning messages will be sent to the Work Center.	Leave blank or use the number 1. If left blank, all warnings and errors are processed.	Blank value
cAllowUserIDToChange	If this feature is turned on, you can, at any time, change the user who receives the messages. Some restrictions do apply.	Leave blank or use the number 1. If left blank, all messages will be sent to one user.	Blank value

szMailboxdesignator	Mailbox (or queue) override. If this field is turned on, then messages are sent to the overridden mailbox. (Work Center use.)	Leave blank or use any valid mailbox designator. If left blank, all messages are written to the default queue.	Blank value
idPPATworkField	The work field used by the Work Center API. This is where the GENLNG work field should be passed.	This must be a work field defined in the UBE. Use the GENLNG data item for this.	The GENLNG work field
cSendErrorsToReport	If this feature is turned on, no messages (except Job Completed) will be sent to the Work Center. The messages will then be available to read through another Work Center function.	Leave blank or use the number 1.	Blank value

3. Choose the F01131 Edit JDEM Error Message function within the API.
4. Refer to the Windows Help file of APIs to identify the parameters to send.

Calling the Processing Work Center APIs

After the Work Center system has been initialized, you must determine the various level-break points within the report and call the Work Center system at each of these points to group the errors.

► To call the processing Work Center APIs

1. From Report Design, establish all of the appropriate level-breaks for the reporting batch job.

You need to analyze the events that will logically group all errors at a given event. This typically happen at events in which all editing has been completed for a group of records or immediately after all editing for an individual record has occurred.
2. For each level-break established, do the following:
 - a. Call the business function for the level-break message at the appropriate level-break.

The business function for the level-break message should relate to the type of error grouping that you want to occur at the particular level-break. For example, SetLevel_SFVoucher groups errors related at the voucher level-break. For reports, this business function is typically called in the Do Section. If the interconnect is blank, then it is not calling an action message.

Note

The level that you send the API is never 1; it is sent only once, when you terminate the process.

- b. Call the Work Center error message function immediately after the call to the level-break message.

This function is called ProcessErrorsToPPAT. B0100011.c processes batch errors to JDEM and makes repeated calls.

- c. Refer to the Windows Help file of APIs to identify the parameters to send.

Use the following table to identify which parameters to send. This is where you designate a level for the data dictionary level-break message. Start your message with level 2. The system automatically generates level 1.

Parameter	Description	Allowed Values	Typical Values
mnLeveloftotaling	The message level that you want the Work Center API to process.	Any valid number from 1 to 20. Use the number 1 only when the UBE is almost finished. 1 writes the Job Completed message.	Numbers 1 to 20
idDataBaseWorkField	A work field used by the Work Center API. The GENLNG work field should be passed here.	Must be a work field defined in the UBE. Use the GENLNG data item.	GENLNG work field
cErrorPreProcessFlag	This parameter allows the UBE to call the Work Center API at some event to flush the error space and to group any errors that are there by a level-break message, which has yet to be sent.	Leave blank or use the letter P. If you leave this parameter blank, errors in the error space are grouped by the level message being sent in the first parameter.	Blank or default value
szUserid	When the cAllowUserIDToChange option is set in the initialization function, this parameter accepts the new user ID.	Leave blank or use any valid user ID. If you leave this parameter blank, all messages are written to the last user ID used.	Blank or default value

cAllowUserIDToChange Parameter

This parameter on the initialize API works in conjunction with the szUserid parameter on the ProcessErrorsToPPAT API. It allows you to set up the UBE so that when any batch errors are encountered, errors are sent to the user who created the original records and not to the person submitting the job (such as the night operator). For example, if a single batch job contains 1,000 transactions created by 50 users, then only those users who created transactions with errors will receive error messages. The night operator will still receive a message, but it would be a message such as "Job completed normally" or "Job completed normally with errors." Other users whose transactions were successful will not receive any error messages.

To set up this functionality, you need to enter a "1" in the `cAllowUserIDToChange` parameter when you initialize the batch error processing system. When you process the level 2 level-break message and then call the `ProcessErrorstoPPAT` API, you can still specify who will receive the messages by using the `szUserid` parameter. You can determine who should receive the message by reviewing the transaction record.

Terminating the Work Center Process

After all messages have been sent to the Work Center, you must terminate the Work Center process before the reporting job is finished. When the batch program is about to terminate, call the Work Center error message function, `ProcessErrorstoPPAT`, one last time, sending it to level 1. (Level 1 indicates the level of totaling is equal to 1 and that it is completed.) The job-completed message is created, freeing any workspace that the Work Center API created.

Every report design that uses the Work Center API to process errors must use level 1 to call the API at the end of processing. This call should also be done by reporting jobs that are monitoring for critical errors and that need to terminate early.

When the system has finished processing the report, it creates Work Center messages, which can then be read using the Work Center application. Batch errors are processed to JDEM system. Messages that are created are sent to the user who ran the report unless you specify that the message be sent to another user.

If the system encounters no errors, the API sends a message to the Work Center to indicate that the job completed successfully.

Debugging

Debugging is the method you use to determine the state of your program at any point of execution. You can use debugging to help you solve problems and to test and confirm program execution.

You can use a debugger to stop program execution so you can see the state of the program at a specific point. This allows you to view the values of input parameters, output parameters, and variables at the specified point. When program execution is stopped, you can review the code line-by-line to check such issues as flow of execution and data integrity.

You can use the following two tools for debugging PeopleSoft EnterpriseOne:

- EnterpriseOne Event Rules Debugger
- Microsoft Visual C++ Debugger

You use the Event Rules Debugger to debug event rules and the following:

- Interactive applications
- Reports
- Table conversions

You use the Visual C++ Debugger to debug C business functions.

Overview of the Debugging Process

You use the debugging process to determine where problems occur and to fix those problems. You should isolate each problem to a particular area, and then examine exactly how the program operates in that area.

If you change your program while you are debugging with the Event Rules Debugger, you must:

8. Disconnect the Event Rules Debugger
9. Rebuild debug information
10. Reset breakpoints
11. Run the application

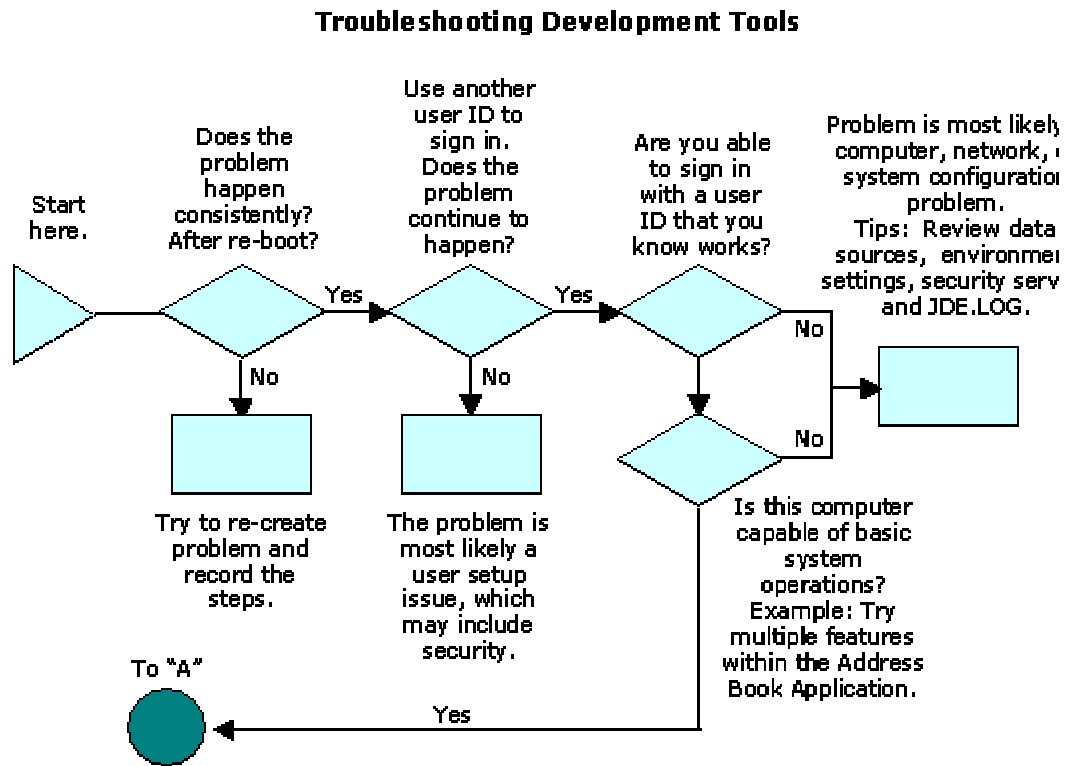
Features available in the debuggers are listed and described in the following table:

Feature	Available in:	Description
Go	Visual C++ and Event Rules Debugger	Command that resumes program execution after a breakpoint is reached. In Visual C++, after you start an application, the application will run until the breakpoint is reached.
Breakpoints	Visual C++ and Event Rules Debugger	Commands that tell the debugger to stop when a particular line is reached. You can set breakpoints on lines of code where you want to start debugging.

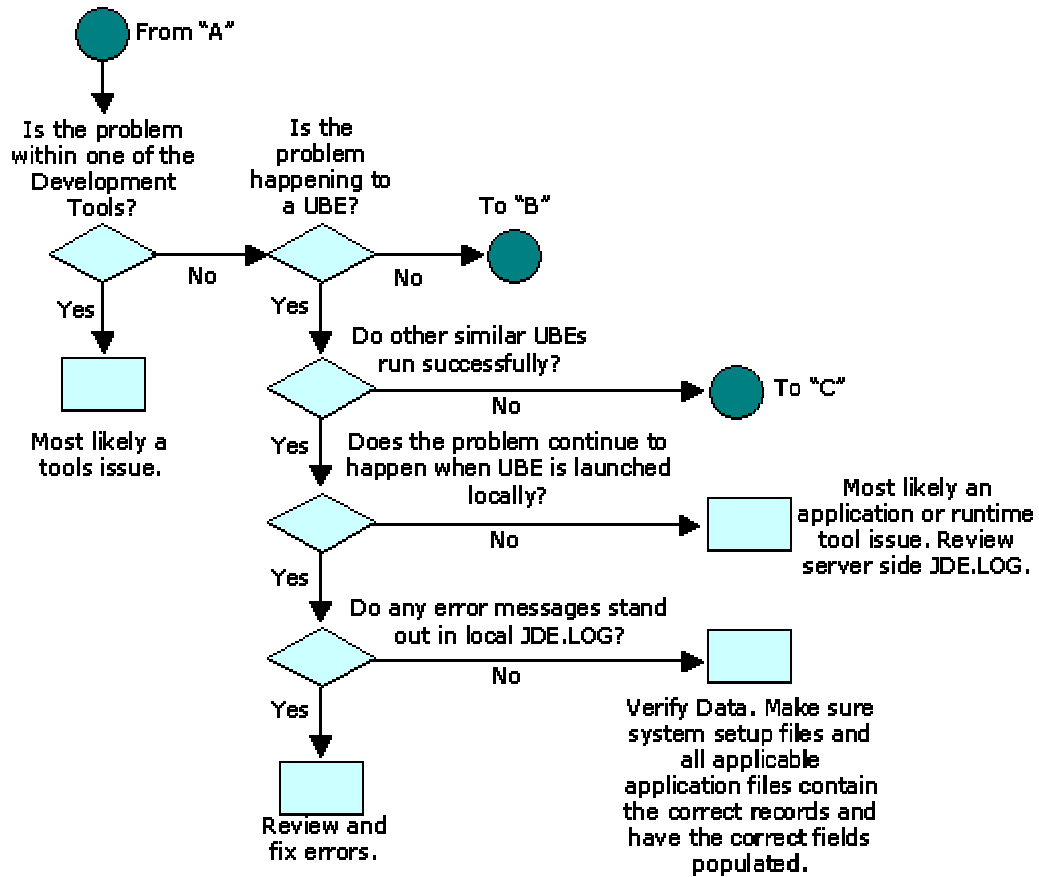
Feature	Available in:	Description
Step, Step Over	Visual C++ and Event Rules Debugger	Command that executes the current line of code. Step lets you run the program one line at a time. You can use this feature to determine the results of every line of code as it is executed.
Step Into	Visual C++ and Event Rules Debugger	Command used when the current line of code contains a function call. The debugger will step into the function so that it can be debugged line by line. When the function is complete, the debugger returns to the next line of code after the function call in the calling routine. Step Into can be used to debug a second application that is called from within an EnterpriseOne-calling application.
Stop	Visual C++ Debugger	Command that stops the debugging process. The running application is also terminated.
Disconnect	Event Rules Debugger	Command that disconnects the debugger from the current application. The application continues to run as if the debugger had not been started.
Watch	Visual C++ and Event Rules Debugger	Command that displays the value of variables while the program is running. It also lets you inspect expressions so that you can see how a particular expression changes when variables change. Both the C++ and Event Rules Debuggers allow you to change the values of watch variables.

Troubleshooting: Development Tools

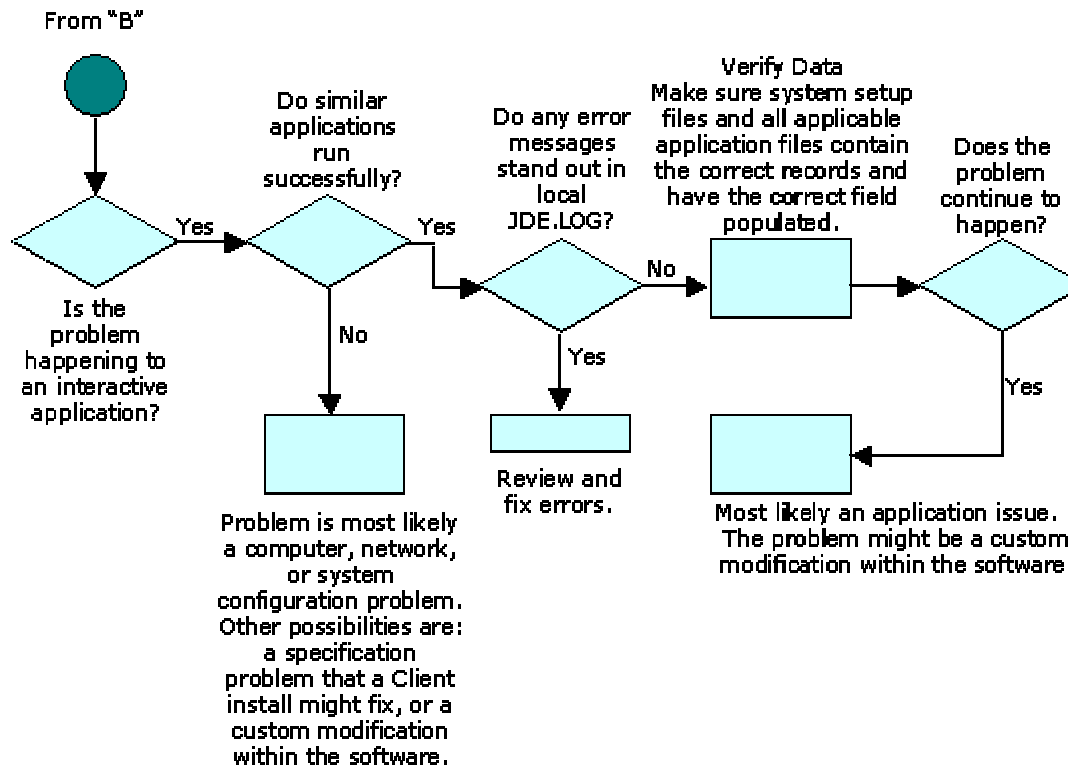
The following graphics show how to determine the cause of problems you might have with development tools:



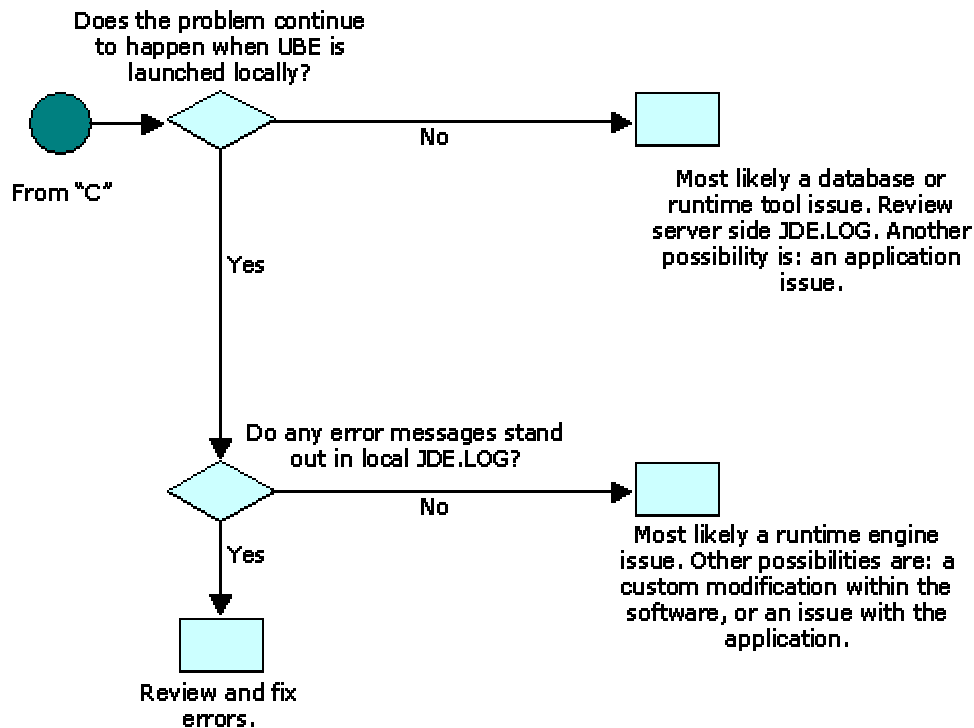
Troubleshooting Development Tools part 2



Troubleshooting Development Tools part 3



Troubleshooting Development Tools part 4



Interpretive and Compiled Code

PeopleSoft EnterpriseOne uses both interpretive and compiled code.

Interpretive code refers to code that is compiled as it runs. The translation from program instruction to machine instruction happens at runtime. Interpretive code resides within the application. Interpretive code allows you to customize applications without having to recompile every time you change something. The scripting code for event rules used in Form Design and Report Design is interpretive.

Compiled code is compiled and stored in an object file that can be called independently. The translation from program instruction to machine instruction happens at compile time. Compiled code resides outside the application. It is less subject to change and is typically less flexible than interpretive code. You can use Visual C++ to debug compiled code. The event rules scripting language can be translated into C, Java, or your current language of choice. The logic only needs to be interpreted one time. Table event rules, named event rules, and business functions are examples of compiled code.

Working with the Event Rules Debugger

The Event Rules Debugger provides the essential debugging tools (such as breakpoints, step commands, and variable inspection) that you need to debug EnterpriseOne interactive and batch applications. You can use the Event Rules Debugger to debug named event rules and table event rules. When the Event Rules Debugger generates debug information for an application, it includes named event rule and table event rule information for that application.

Setting up and using Event Rules Debugger involves two steps:

- Running the Event Rules Debugger and setting breakpoints
- Running and debugging the application, report, or table conversion

If you want to save the debug information table but do not want the Debugger active during your work, you can deactivate debug information. You can activate the table at any time to continue debugging.

Understanding the Event Rules Debugger

The Event Rules Debugger is a stand-alone tools program that consists of five main controls:

- Object Browse window
- Event Rules window
- Breakpoint Manager
- Variable Selection and Display window
- Search combo box

All windows except the Event Rules window are dockable to any side of the main application. You can right-click a window to dock it or to hide it. If you close the Debugger, it saves your docking settings until the next time you run the Debugger.

Object Browse Window

The Object Browse window lists applications that have debug information built and are available for debugging. You can navigate through a tree structure to a specific event and open an Event Rules window for that event. If one of the objects is a Power Form with subforms, the subforms are listed under the Power Form. Form IDs appear next to the form name.

Event Rules Window

The Event Rules window displays the event rules for one event. The event name and path, along with an abbreviated description, appear in the title bar for each Event Rule window. The Event Rules window shows the line in the event rules that is currently being executed when the runtime engine is stopped on a line break.

The left side of an Event Rule window displays icons that describe the state of a line in the event rules. States include the following:

- Breakpoint
- Disabled
- Current line of execution

You can use the Event Rules window to set and remove breakpoints. You can use any of the following methods:

- Double-click the line in event rules.
- Choose a line and choose Breakpoints from the Debug menu.
- Right-click a line and choose Breakpoints from the pop-up menu.
- Choose a line and press F9.

You cannot set a breakpoint on a comment. The breakpoint automatically goes to the first code line below the comment.

Variable Tree and Watch Window

The Variable Tree and Watch window consists of two panes. The left pane is the Variable Tree pane. It contains a tree structure that lists the variable types as parent nodes and the current variables of each variable type as child nodes. The variables displayed are those that are in scope of the currently displayed event in the Event Rule window. The right pane is the Watch pane. It displays user-selected variables and their current values, along with the variable's form ID. You can add a variable to the Watch pane by double-clicking the desired variable in the Variable Tree.

You can change the value of variables while you are debugging an application. To change the value of a variable, double-click the variable in the Watch pane and enter a different value. The new value appears in the Watch pane. If you enter an inappropriate value (for example, you change a numeric value to an alpha value) the new value is not set and the value is not changed.

The following three special values are displayed for variables:

- blank** The value for the variable contains only blanks. This value applies to string and character types only.
- null** The variable has no value, or a null or empty value.
- unknown** The value for the variable could not be obtained from the runtime engine. This value occurs when the applications are not running or when the variables are out of scope.

Note

Variable inspection and modification is not available for debugging named event rules and table event rules.

Breakpoint Manager

Breakpoints tell the debugger where or when to halt the execution of a program. When the program is halted at a breakpoint, you can examine the state of your runtime structures, review your event rules, and evaluate expressions using the Variable Selection and Display window.

The Breakpoint Manager tracks the breakpoints that are set and the location of those breakpoints in an application. When you set a new breakpoint, the system creates an entry in the Breakpoint Manager. This entry contains the application name, form name, event name, and event rule line.

Right-click within Breakpoint Manager to perform the following operations:

- Delete a breakpoint.
- Delete all breakpoints.
- Go to a breakpoint.

You can also double-click an entry in Breakpoint Manager to open the Event Rule window in which the breakpoint is set.

Search Combo Box

You can use the Search combo box on the toolbar to search for event rule text. Enter the text that you want to search for in the Search combo box and then press either Enter or F3. If the system locates the search text in your event rules text, it highlights the text. If you press Enter or F3 again, the next occurrence of your search text is located and highlighted.

The search control accommodates regular expression searches. A regular expression search uses special characters to match text. For example, `^If:` will find every line that starts with "If" and `If$` will find every line that ends with "If."

The special characters that you can use for advanced searches are described in the following table:

^	The caret (^) indicates the beginning of a string. For example, the expression " <code>^A</code> " matches an "A" only at the beginning of the string.
^	The caret (^) immediately following the left bracket ([) is used to exclude any remaining characters within brackets from matching the target string. For example, the expression " <code>[^0-9]</code> " indicates that the target character should not be a digit.
\$	The dollar sign (\$) matches the end of the string. For example, the expression " <code>abc\$</code> " will match the substring "abc" only if it is at the end of the string.
	The alternation character () allows the expression on either side of it to match the target string. For example, the expression " <code>a b</code> " will match "a" as well as "b."
.	The dot (.) matches any character.
*	The asterisk (*) indicates that the character to the left of the asterisk in the expression should match 0 or more times.
+	The plus (+) is similar to the asterisk except that at least one match of the character should occur to the left of the + sign in the expression.

- ? The question mark (?) matches the character to its left 0 or 1 times.
- () The parentheses affect the order of pattern evaluation and serve as a tagged expression that you can use to replace a matched substring with another expression.
- [] Brackets that enclose a set of characters indicate that any of the enclosed characters can match the target character.

Debugging an Application

Debugging is the method you use to determine the state of your program at any point of execution. You can use debugging to help you solve problems and to test and confirm program execution. The Event Rules Debugger allows you to debug interactive or batch applications.

► To debug an application

From the Cross Application Development Tools menu (GH902), choose Debug Application.

1. Choose the object that you want to debug.
2. Choose a form (for interactive applications) or section (for batch applications) and an event to view.
3. Choose the event rule line on which you want to set a breakpoint.
4. From the Debug menu, choose Breakpoint.

A red dot appears on the line, indicating the breakpoint.

You can remove the breakpoint by choosing Breakpoint from the Debug menu. The options on the Debug menu toggle on and off.

5. Minimize, but do not close, Debugger.
6. From PeopleSoft EnterpriseOne Menus or from the Object Librarian, run the application that you selected to debug.

As your application encounters a breakpoint, the application will pause and display the Event Rules Debugger.

When execution stops, you can use the variables view to inspect and modify the values of runtime structures.

7. From the Debug menu, choose one of the following options:
 - Go
 - Disconnect
 - Step Over
 - Step Into

Inspecting or Modifying a Variable

When you debug an application and encounter a point at which the interactive or batch application fails, you can view the appropriate variable and correct it.

By observing specific variables while the program runs, you can isolate where the program begins to fail and what exactly it is doing. For example, if a counter is supposed to increment by 1, but you observe it incrementing by random numbers, you know there is a problem with the number or variable you are adding to the counter.

► To modify a variable

1. On Event Rules Debugger, double-click a variable in the Event Rules Variables window.
2. Revise the information in the following field:
 - New Value

Setting Breakpoints

Any event that you want the Debugger to stop on must have at least one line of event rule code. You cannot set a breakpoint on a comment. It automatically goes to the first code line below the comment.

Debugging Business Functions By Using Microsoft Visual C++

You can use Microsoft Visual C++ to debug business functions that are written in C. You must use version 6.0 Service Pack 5 of Microsoft Visual C++. You can debug business functions that are attached to interactive applications or to batch applications. The business function must be configured to run locally.

Debugging Business Functions Attached to Interactive Applications

You can debug business functions that are attached to interactive applications or to batch applications.

► To debug a business function attached to an interactive application

1. Close the application.

The application must be closed to debug in Visual C++.
2. Open Visual C++ and verify that all workspaces have been closed.
3. From the File menu, choose Open.
4. Choose List Files of Type to accept executables (.exe).
5. Select activConsole.exe on path \b9\System\bin32 and click OK.

The system creates a project workspace.

6. From the Project menu, choose Settings.
7. Click the Debug tab.
8. In the Category list, choose Additional DLLs.
9. Click the Browse button to select the CALLBSFN.dll or other appropriate DLL on path `\b9\path\bin32`, where *path* varies, depending on the path code.
10. Click OK.
11. Choose your .h and .c files for the source that you want to debug from and then, from the File menu, choose Open.
12. To set breakpoints in your code, choose Breakpoints from the Edit menu

If the following message appears, click OK:

```
cannot open *.pdb
```

If a message appears notifying you that breakpoints have been moved to the next valid lines, a source code and object mismatch might exist, and you might need to rebuild your business function.

13. From the Build menu, choose Start Debug, Go.

The PeopleSoft EnterpriseOne sign-in window appears.

14. Sign on to the application as you normally would sign on.

15. Run the application.

When your application reaches the business function in Debug, the debugger opens or displays the C code in Visual C so that you can step through it.

If you are debugging event rules for business functions and C business functions, you can use the EnterpriseOne debugger and the Visual C++ debugger together. Follow the steps above until you log into EnterpriseOne. At that point, follow the steps for the EnterpriseOne debugger. Program execution stops if C code is accessed. You can then use Visual C++ to continue debugging. This method is useful if you are trying to locate a problem and are not sure whether the problem is in a C business function or in the application that calls the business function.

Debugging Business Functions Attached to Batch Applications

You can debug business functions that are attached to batch applications.

► To debug a business function attached to a batch application

1. Close the PeopleSoft EnterpriseOne application.
2. Open Visual C++ and verify that all workspaces are closed.
3. From the File menu, choose Open.
4. Specify executables (.exe) in the Files of Type field.
5. Locate and choose LaunchUBE.exe

6. On LaunchUBE, browse for the report and version that you want to use.
7. Complete the Server Name field by specifying the location in which the report will run.

You can choose Local or specify a server.

8. Set the Debug Level.
9. Click one of the following Print Options:
 - On Screen
 - To Printer
10. Click Submit to submit the report.

Working with the Visual C++ Debugger

You must use the Microsoft Visual C++ Debugger to debug business functions written with the Event Rules scripting language or in C code. You can run the entire EnterpriseOne system through the Visual C++ debugger (that is, you can start the activeConsole.exe or PeopleSoft Solution Explorer file from within the Visual C++ Debugger). This allows you to "step out" of the CASE tool-generated application code into the business functions that are called in the event rules.

You can use the debugger to debug a C program and interactively stop and start it as needed. During debugging, you can check specific values of variables and parameters to determine whether a program is running correctly. You can also step through the code to see what code is actually being executed.

The debug commands are listed in the Debug menu. You can customize the toolbar to contain debug buttons, which you can use instead of the menu.

Useful Features of the Visual C++ Debugger

The Visual C++ has many features in the Debug menu. The Visual C++ debugger helps you efficiently solve real-world problems.

The Go Command

You can run a program using the Go command from the Debug menu. The program runs until completion unless you set up breakpoints.

The Step Command

The Step command is available on the Debug menu and executes the current line of code. When the line of code has been executed, the yellow arrow cursor appears on the next line of code to be executed.

The Step Into Command

You can access the Step Into command from the Debug menu. Use this command when the current line of code contains a function call. The debugger steps into the function so that it can be debugged

line by line. When the function is complete, the debugger returns to the next line of code after the function call in the calling routine. If the source code of the function to be stepped into does not exist on the workstation, the debugger skips over the line of code as though the Step command was used.

Stepping into a standard C function takes you into the function, which you might not want to do. If so, use the Step Over command to skip those functions.

Setting Breakpoints

You use breakpoints to run the program until it reaches a certain line of code. If a breakpoint is set, the Go command runs the program until it encounters that line of code.

You can set a breakpoint by placing the cursor anywhere on the line of code. When you choose Breakpoints from the Debug menu, a red octagon appears to the left of the line of code where the breakpoint is set. When the program is run, all lines of code up to the breakpoint are executed. To continue execution after the breakpoint, you can use Step, Step Into, or Go.

Using Watch

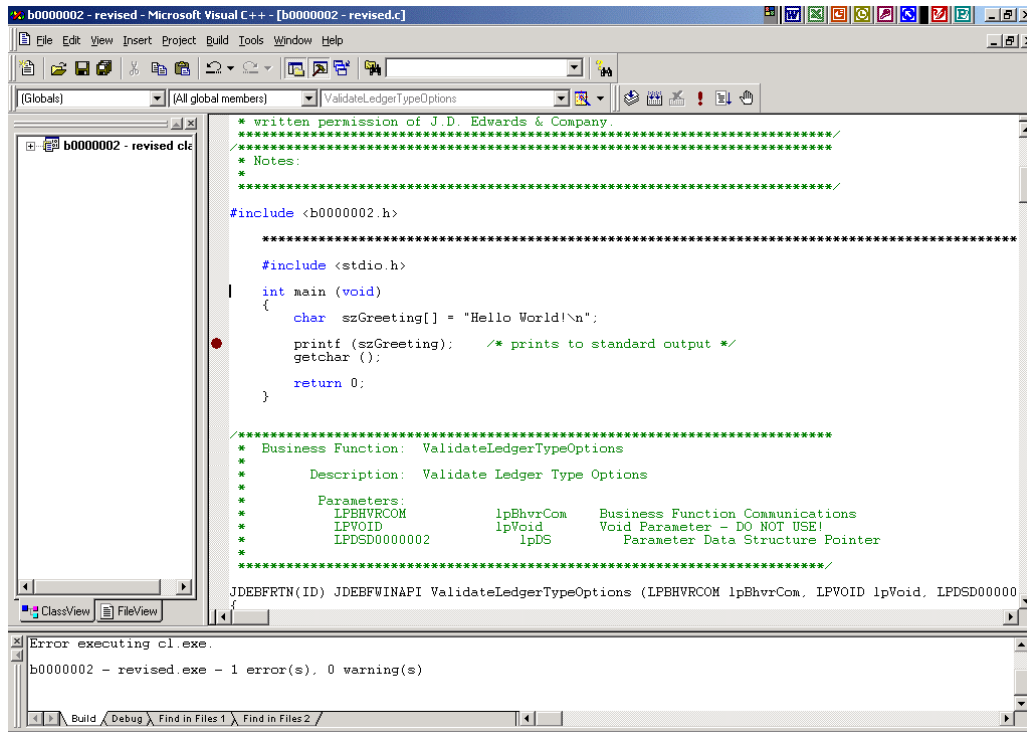
You can use Watch to inspect what values variables are set to. To use Watch, click the item that you want to watch and drag it to the Watch window.

Locals Window

All local variables and parameters to a function are listed with their data types and values in the Locals window. You can modify the values of all items in the Locals window during debugging. This is useful if you are debugging infinite loops.

Example: Debugging with the Visual C++ Debugger

In the following example, a breakpoint, Watch, and Locals have been used.



- A breakpoint has been set on the line of code with the call to `printf`. The red octagon is to the left of the code. The call to `printf` has been executed with the Step command.
- The current line to execute has a yellow arrow to the left of it. It contains a call to the `getchar` function. That line of code has not yet been executed.
- `szGreeting` appears in the Locals window and in the Watch window. You can scroll these windows horizontally to review the entire line. You can see the beginning of the value of `szGreeting` in the Watch window.
- A plus sign next to a variable in the Locals and Watch windows indicates the data may be expanded. This is useful for inspecting the values of complex data structures.

Customizing the Environment

You can modify the Visual C++ debugging environment. You can customize the editor, colors, fonts, debugger, default directories, workspace, and help.

► To customize the environment

1. On Event Rules Debugger, from the Tools menu, select Options.
2. On Options, customize your environment as desired.

Debugging Strategies

You can use several strategies to make debugging faster and easier. Begin by observing the nature of the problem.

Is the Program Ending Unexpectedly?

If the program is ending unexpectedly, the cause is likely an unhandled exception. An unhandled exception is a failure to handle memory correctly. It is an easy problem to track down if it is happening in the same place: simply set breakpoints at strategic points throughout the code and run the program until you find the problem.

If the problem resides in C code, you can find the problem by tracing into the code.

If the problem exists in the EnterpriseOne-generated code, it might be more difficult to find. The debugger provides error messages that are helpful. The most common problem has to do with missing objects. If a business function that is being called does not exist, the tool issues an error message that identifies the missing function. For example, "Business function load failed - CALLBSFN.DLL." In this case, you can either rebuild the business function or check it out and build it using BusBuild to correct this error.

Remember that *all* business functions are built into larger DLLs. The most generic of these is CALLBSFN.DLL. Most application-specific DLLs are not in CALLBSFN. For example, EnterpriseOne financial business functions use CFIN.DLL. The object (BSN func) might need to be checked out to the workstation again and built through BusBuild again in CALLBSFN.DLL (or the specific DLL).

If other objects are missing, termination is more abrupt. Remember to transfer all Media Object (also called Generic Text) objects correctly. If an application has a Row exit to an application that does not exist, an unhandled exception in the program occurs immediately.

Termination of the program is more abrupt and less helpful when other kinds of objects are missing. You must review all of the pieces of your application to verify that they are all present and correctly built. A common error is to overlook media objects. If you cannot enter your program at all, a missing object is most likely the problem.

Ensure that the program is terminating in the same place. If the program is failing to restore memory after its use, the program might eventually have insufficient memory to run. If so, you must reboot the workstation to restore memory.

Is the Application Encountering Errors?

When a business function is called from event rules, its processing is unknown to the user. This situation is both a benefit and a hindrance.

- It is a benefit because it should not be a concern to the user how the function works, only that it does work.
- The disadvantage is that it is not uncommon to call a function with more than 50 parameters, any one of which might cause errors.

The two solutions to this sort of issue are:

- Review the function specifications to make sure that you have hooked it up correctly.
- Step into the code to see what is going wrong.

Is the Output of the Program Incorrect?

Incorrect program output indicates a flaw within the logic of the code. Trace into the code to find the issue.

Where Else Could the Problem Be Coming From?

Spend some time thinking about where the source of the problem might be.

For example, consider the "Null pointer error." Somewhere in the code memory is a memory leak. To help determine where the leak is, you can toggle between the grid rows to force the execution of the `Row is Exited` event. If the problem occurs, you have narrowed your search for the leak, because you know that it is occurring in the grid processing.

Is the Function Being Called as Expected?

You can set a breakpoint at the beginning of a called function to ensure that it is being called and that the input is as you expected. To do so, set the breakpoint at the beginning of the business function or at the beginning of an event such as `Grid Row is Fetched`, and then do the following:

- Step through every code path.
Use the debugger to test the code. Both the PeopleSoft EnterpriseOne Debugger and the Microsoft Visual C++ Debugger provide you with the ability to modify values as the code is running. Use this ability to trace through *every* line of code. This is also a powerful way to test everything that could happen to your code, not just what *should* happen to it. This method of testing will help demonstrate whether you have handled errors correctly. For example, step through an If branch and input bad data to see if errors occur as expected.
- Find the cause rather than the symptom of the problem.
- Look for problems with object transfer/reinstall.

Debug Logs

Several sections of the `jde.ini` file relate to debugging. The first statement that must be checked is in the `[JDE_CG]` section. The line `Target = Release` should be modified to read `Target = Debug`, as shown below. An application install of ERP will deliver the `jde.ini` file with `Target = Release`. To debug business functions, the `jde.ini` files must be changed to `Target = Debug`, and the business functions that you want to debug must be rebuilt.

```
[JDE_CG]
STDLIBDIR=c:\MSDEV\LIB
TPLNAME=EXEFORM2
```

```
ERRNAME=CGERR
TARGET=Debug

INCLUDES=c:\MSDEV\INCLUDE;$ (SYSTEM) \INCLUDE;$ (SYSTEM) \INCLUDEDEV;
$ (SYSTEM) \CG;$ (APP) \INCLUDE;

LIBS=c:\MSDEV\LIB;$ (SYSTEM) \LIB32;$ (SYSTEM) \LIBV32;$ (APP) \LIB32;

MAKEDIR=c:\MSDEV\BIN

USER=DEMO
```

You can output to a file a log of SQL statements and events by changing the line in your jde.ini file under [DEBUG] from `Output = NONE` to `Output = FILE`, as shown below. This is a useful debugging tool when you have narrowed a problem to a specific issue involving the JDEDB APIs.

```
[DEBUG]

TAMMULTIUSERON=0

Output=FILE

ServerLog=0

LEVEL=BSFN,EVENTS

DebugFile=c:\jdedebug.log

JobFile=c:\jde.log

Frequency=10000

RepTrace=0
```

The memory frequency setting allows you to validate the memory heap at a particular 1000th location. You can set breakpoints and examine the code.

See Also

- ❑ *Viewing Log and Debug Log Files for Enterprise Server Processes in the Server and Workstation Administration Guide* for more information about other logs that you can view for troubleshooting

SQL Log Tracing

You can use SQL Log Tracing to help you determine the exact SQL statement that is generated and sent to the database.

► To turn on SQL Log tracing

1. From the Control Panel on your workstation, choose Administrative Tools, and then Data Sources (ODBC).
2. Choose the 32 bit ODBC driver, and then click the Tracing tab.
3. Specify when you want the system to trace.
4. Specify the log output path in the Log file Path.

Debug Tracing

Use debug tracing to trace database and runtime events, business functions, and system functions. You can turn on debug tracing and store the results in a log file. This debug log is also useful for verifying how the tool constructs an SQL statement.

► To turn on debug tracing

1. In the JDE.INI file under [DEBUG], change Output=NONE to one of the following values:

Output=FILE Prints both database tracing and runtime tracing.

Output=EXCFILE Prints runtime tracing only.

2. Change the value for Level= to suit your specific debugging needs.

Possible values for Level are contained in the comment line following the Level= line. Any combination is acceptable. Use commas to separate values. The possible values for Level are as follows:

EVENTS Records when an event starts and when it finishes.

BSFN Records when business functions are entered and when they return.

SF_? Records when system functions execute. In place of the question mark, you can designate a specific type of system function, such as control or messaging.

System Function Tracing

When you review the jdedebug.log, note that system functions are grouped by category (using the same categories as design time). Where reasonable, the log includes information about what component within the program the system function is acting upon. For example, in many of the grid system functions the row number is printed. For most control system functions the log includes the alias of the control and the control ID.

The following example shows the jdedebug.log output as a result of running the Journal Entry with the following jde.ini options:

```
Output=EXCFILE
```

```
Level=EVENTS,BSFN,SF_GRID,SF_CONTROL
```

RT: >>>Beginning ER: Dialog is Initialized App: P0911 Form: W0911I
 RT: <<<Finished ER: Dialog is Initialized App: P0911 Form: W0911I
 RT: >>>Beginning ER: Post Dialog is Initialized App: P0911 Form: W0911I
 RT: <<<Finished ER: Post Dialog is Initialized App: P0911 Form: W0911I
 RT: >>>Beginning ER: Add Button Clicked App: P0911 Form: W0911I
 RT: >>>Beginning ER: Dialog is Initialized App: P0911 Form: W0911A
 RT: SYSFN: Hide Control <> 0
 RT: SYSFN: Disable Control <ICU> 5258
 RT: SYSFN: Hide Grid Column COL: 5
 RT: SYSFN: Hide Control <ATDOW> 5392
 RT: SYSFN: Hide Control <REMA> 5405
 RT: SYSFN: Hide Control <> 5295
 RT: SYSFN: Hide Control <> 5385
 RT: SYSFN: Hide Control <DOC> 5297
 RT: SYSFN: Hide Control <KCO> 5299
 RT: SYSFN: Hide Grid Column COL: 7
 RT: SYSFN: Hide Grid Column COL: 8
 RT: SYSFN: Hide Grid Column COL: 9
 RT: SYSFN: Hide Grid Column COL: 11
 RT: BSFN: Calling : BatchOpenOnInitialization App: P0911 Form:
 W0911A
 RT: BSFN: Returned 0: BatchOpenOnInitialization App: P0911 Form:
 W0911A
 RT: BSFN: Calling : GetAuditInfo App: P0911 Form: W0911A
 RT: BSFN: Returned 0: GetAuditInfo App: P0911 Form: W0911A
 RT: <<<Finished ER: Dialog is Initialized App: P0911 Form: W0911A
 RT: >>>Beginning ER: Clear Screen Before Add App: P0911 Form: W0911A
 RT: SYSFN: Enable Control <PCTOW> 5390
 RT: SYSFN: Hide Control <ATDOW> 5392
 RT: SYSFN: Hide Grid Column COL: 5
 RT: SYSFN: Hide Control <REMA> 5405
 RT: SYSFN: Enable Control <CRCD> 5273
 RT: SYSFN: Enable Control <LT> 5292
 RT: SYSFN: Enable Control <LT> 5351
 RT: SYSFN: Show Grid Column COL: 6
 RT: SYSFN: Hide Grid Column COL: 12
 RT: SYSFN: Show Control <LT> 5292
 RT: SYSFN: Show Control <CRDC> 5271
 RT: SYSFN: Hide Control <LT> 5351
 RT: SYSFN: Hide Control <CCD0> 5358
 RT: BSFN: Calling : GetLocalComputerId App: P0911 Form: W0911A
 RT: BSFN: Returned 0: GetLocalComputerId App: P0911 Form: W0911A
 RT: <<<Finished ER: Clear Screen Before Add App: P0911 Form: W0911A
 RT: >>>Beginning ER: Post Dialog is Initialized App: P0911 Form: W0911A
 RT: <<<Finished ER: Post Dialog is Initialized App: P0911 Form: W0911A
 RT: >>>Beginning ER: Add Last Entry Row to Grid App: P0911 Form:
 W0911A
 RT: <<<Finished ER: Add Last Entry Row to Grid App: P0911 Form:
 W0911A

Developing Web Applications

Tools make it easy for you to develop and maintain applications for use on both Windows clients and Web clients, even if the clients require slight differences in form design. You can create Windows, Java, or HTML applications. As you design a form, Form Design Aid allows you to view it in one of three modes. Your design team should decide which modes to use for which client types. For example, you might use Mode 1 for forms to be generated in Windows, Mode 2 for forms to be generated in Java, and Mode 3 for forms to be generated in HTML.

Most of the controls that you put on a form are appropriate for any of the modes, but you can choose to show or hide (enable or disable) any control on the applicable form for each of these three modes. In this way, you can develop one set of forms from which you generate three versions of the application (one for each mode). If you customize any of the EnterpriseOne Web applications or if you develop new ones, you must use the Web Generation Facility to generate them into Java or HTML applications.

Typically, you create an application in Mode 1, which is the default mode. This application is automatically enabled for Windows. You can then use Mode 2 or Mode 3 to modify your application for Java or HTML.

Although you can develop or change Web-based applications in Java or HTML on your workstation, you need a workstation that is Web-enabled to test the application. All EnterpriseOne clients are Web-enabled. The following components are required:

- Internet Explorer 4.01 or above
- EnterpriseOne Web Generation Facility (installed with EnterpriseOne client)
- Network access to an EnterpriseOne Web server

If you customize any of the EnterpriseOne Web applications or if you create new applications, you must regenerate them to create the specified Java or HTML applications. You must generate to create Java or HTML applications.

You might want to make changes in a Windows application to make it easier to use on the Web. For example, you typically use fewer fields in a Web application.

Note

If you change the font size in your style sheets, be aware that doing so can alter the appearance of the form, including displacing controls and wrapping or hiding control text.

To test a Java application, you must do the following:

1. Save your application in Form Design.
2. Generate the application to the Web server.
3. Attach the application to a menu and designate the mode in which it should run.
4. Run the application.

To create Web applications, use Tools in the same way would use it for standard applications.

If you use Tools to create your own applications and these applications are designed for use as Web applications, you should consider the following to ensure optimal performance:

- Each data and business function request runs on a data or enterprise server, not the Web server. As a result, your Web client performance is limited by the speed of your network or modem connection, as well as network traffic restrictions. If possible, you should limit or bundle data and business function requests in a manner most efficient for WAN and Internet connections. If you have several table I/O calls serially linked (for example, each one is dependent on the previous call), consider using named event rules to combine all calls, which are compiled and executed on the enterprise server.
- Whenever possible, design the Web-ready application to perform logic using event rules. This design provides better performance than if you use C language business functions or event rules business functions (named event rules). Keep in mind that whenever you call a business function the logic is performed on the remote EnterpriseOne enterprise server. However, if the event rules contain several table I/O or business function calls that are serially linked, these will perform better as a business function (named event rule) on the enterprise server.
- Design the Web-ready application to request data in manageable chunks. If your application performs a business function request before and after each grid line, performance will suffer. In addition, avoid using business functions as a means to request data; instead, use business views or table I/O.
- Educate users in the importance of using search criteria in query by example columns. This is important given the constraints of transferring data at modem speeds across typical Internet connections. For example, users should routinely try to narrow their searches, because data retrievals are configured to obtain only 10 records at a time. If you develop custom applications for Web use, consider defining query by example fields as required entries.

Batch Versions - Web Version (P98305W) is a version list application created specifically for the Web client. You can use this application to look up Web versions of applications.

The HTML Client

The HTML client automatically posts (refreshes) on the following critical events:

- Form startup
- Set focus on grid (only when ER exists)
- Check box (only when ER exists)
- Radio button (only when ER exists)
- Bitmap clicked
- URL clicked
- Tab is selected
- Node expanded in tree control
- Drag-and-drop action occurs in tree control

In addition to the critical events listed above, the client also refreshes the page when non-critical events occur that contain hide/show or enable/disable grid functions. The following is a list of these non-critical events:

For Edit controls	Control is exited.
	Control is exited and changed-inline.
	Control is exited and changed-asynchronous.
For Grid controls	Column is exited.
	Column is exited and changed-inline.
	Column is exited and changed-asynchronous.
For Parent/Child form grid	Tree node level is changed.
For Tree control	Tree node is selected.

These post rules for critical and non-critical events form the default post model used by the HTML client. The HTML post model can cause usability issues for high volume data entry applications and inquiry applications that are used frequently on the Web. The issue occurs when an application performs hide/show or enable/disable grid on non-critical events, which forces the HTML page to refresh. The refresh typically causes the screen to flash. The duration of the flash depends on network traffic, hardware, system configuration, and so on. Consequently, you might want to override the HTML post model in some situations.

You can use the HTML Auto Refresh option in a form's property settings to turn off the automatic post on all non-critical events for that form (the default is to have the attribute turned on). Use the HTMLPOST option in an event rule to turn on the post for a single non-critical event (the default is to have the attribute turned off). Enabling posting overrides a disable. For example, if you turn off posting for a form but turn it on in the ER for a single event, posting is suppressed for all events on the form except for the one with the HTMLPOST option turned on in its ER.

The Windows and Java clients ignore both the form and event attributes.

Prerequisite

- ❑ Install the HTML Web Server.

See Also

See the following topics in the *Form Design Aid Guide*:

- ❑ *Designing Forms Using Multiple Modes* for information about HTML form design
- ❑ *Form Controls* for more information about using different modes and customizing the controls on forms for different modes

Performance

This section includes tips and tools to help you enhance the performance of your applications.

Triggers

Data dictionary triggers can be used for both formatting and validation. Triggers are the most costly way to format or edit data values. Most of the formatting that occurs in EnterpriseOne involves date and math fields. These fields require the most overhead for editing and formatting. Ensure that all triggers are coded as efficiently as possible.

Overrides

You can use data dictionary overrides to override the data dictionary characteristics of a form control, grid column, or report field at design time. This causes additional overhead at application startup because of the extra processing required to evaluate and apply the overrides. The overrides that are applied can have either a positive or negative affect on performance during runtime processing. For example, if you use the override feature to disable some of the validation functionality, then performance is increased because disabling features causes some runtime overhead to be removed. Adding overrides such as edit/format triggers or next numbering increases runtime overhead and decreases performance.

Validation

Validation is based on the data dictionary item that is attached to a control, column, or field. Overrides can also affect validation. Validation is most costly for data items that have triggers. Validation of user defined codes requires I/O to validate data values. General validation can be more costly for data types that require special logic to manipulate the EnterpriseOne-specific data types.

Table Design Performance

Use SELECT statements cautiously. Try to use existing indices for a table. Using additional keys is better than creating a new index. Additional indices create overhead.

Use a partial key only for sequential and fetch next or to review a few more records.

If you open a table and then a fetch key, the action destroys the pointer, so perform a fetch single instead. Almost any other database API also destroys the pointer.

Opening a table the first time is a big performance drain.

The fetch matching key uses a greater than or equal key so it might select more than you need. Use the correct JDB API for what you need.

Index Considerations

Adding a suitable index will almost always improve performance when the system selects and fetches data from the database. However, each additional index adds maintenance overhead when records are added, updated, or deleted in the database. You should consider these two factors when you weigh the decision to add a new database index over a table.

EnterpriseOne is designed to access databases efficiently. Use ordinary database design considerations, including normalization considerations, when determining the optimal design of database tables and indices.

Join fields should be the keys in two tables.

Limits on Row Set Size

The lowest common denominator for row size is the specification given in the SQLSERVER database.

SQL Server 6.5 can have as many as 2 billion tables per database and 250 columns per table. The maximum number of bytes per row is 1962. If you create tables with `varchar` or `varbinary` columns whose total defined width exceeds 1962 bytes, the system allows the table to be created; but it generates a warning message.

If you insert more than 1962 bytes into such a row or update a row so that its total row size exceeds 1962, the system generates an error message and the statement fails.

Index Limitations for Various Databases

This section discusses index limitations for the following databases:

- Access32
- SQLSERVER
- DB2 for OS/400
- Oracle

Access32

The following index limitations apply to Access 32:

- The maximum number of indices per table is 32.
- The maximum number of fields in an index is 10.
- The maximum number of fields in a record is 255.

SQLSERVER

The following index limitations apply to SQLSERVER:

- The maximum number of clustered indices per table is one.
- The maximum number of nonclustered indices per table is 249.
- The maximum number of columns in a composite index is 16.

DB2 for OS/400

The following index limitations apply to DB2 for OS/400:

- The maximum number of identifiers for an index name is 10.
- The maximum size of an index is one terabyte.
- The maximum length of an index key is 2,000.
- The maximum number of columns per index key is 120.
- The maximum number of indexes per table is approximately 4,000.

Oracle

The following index limitations apply to Oracle:

- The maximum length for an index is 254 bytes, minus the number of keys that allow NULL values.
- The maximum number of columns per index is 16, or a maximum key length of 900.

Example: Table Index

In the F32944 table, the index is defined as follows:

```
{
  MATH_NUMERIC ktkit;          /* 0 to 48 */
  char   ktmc01[251];          /* 49 to 299 */
  char   ktmc02[[252];         /* 300 to 550 */
  char   ktmc03[251];          /* 551 to 801 */
  char   ktmc04[251];          /* 802 to 1052 */
  MATH_NUMERIC ktseqn;         /* 1053 TO 1101 */
} KEY1_F32944, FAR *LPKEY1_F32944;
#define ID_F32944_KIT_CONFIGURED_STRING_ID_B 2L
```

This situation is not permitted because the key length, in this case, is 1101 (>900).

Key Column Violation

When you define indexes on various columns, ensure that no two indices, one of which might be a primary unique index, are defined on the same column.

Example: Key Column Violation

In this example, the indexes for the Credit/Collection Date Pattern table (F03B08) were defined as follows:

```
#define ID_F03B08_COMPANY_FISCAL_YEAR 1L /* PRIMARY & UNIQUE */
typedef struct
{
    char rdco[6];          /* 0 to 5 (ASC) */
    MATH_NUMERIC rdctry;   /* 6 to 54 (DESC) */
    MATH_NUMERIC rdfy;     /* 55 to 103 (DESC) */
    MATH_NUMERIC rdpn;     /* 104 to 152 (DESC) */
} KEY1_F03F08, FAR *LPKEY1_F03B08;
#define ID_F03B08_COMPANY_FISCAL_YEAR_(ASCEND) 2L /* NONUNIQUE */
typedef struct
{
    char rdco[6];          /* 153 to 158 (ASC) */
    MATH_NUMERIC rdctry    /* 159 to 207 (ASC) */
    MATH_NUMERIC rdfy;     /* 208 to 256 (ASC) */
    MATH_NUMERIC rdpn;     /* 257 to 305 (ASC) */
} KEY2_F03B08, FAR *LPKEY2_F03B08;
```

This results in a Key Column violation, which is not permitted because two indexes have been defined on the same column.

Specification File Corruption

If you encounter specification file corruption, re-create the tables in the source database.

Table I/O Objects

The following guidelines apply to table I/O objects:

- Table I/O header size is 206 bytes.
- Table I/O mapped item size is 181 bytes.
- Maximum number of table I/O mapped items size to fit in 32K is $(32768-206)/181$ for about 180 items.

To provide space for literal values, use a mapping size not greater than 130 elements. The 130 elements relate to the number of mappings on any table I/O statement. A table can have more than 180 columns, but you can still map fewer than 180 of them without problems. If you need to use table I/O with more than 180 mappings, PeopleSoft recommends that you create several table I/O statements for specific requirements.

Business View Performance

This section discusses the performance ramifications of using single-table or multiple-table business views, restricting the number of fields in a business view, unions, and joined business views.

Should I Use a Single-Table or Multiple-Table Business View?

In many instances, an application must access data from multiple database tables to perform a business operation. You can accomplish this in two ways:

- Use a multiple-table business view to access all the related data fields.
- Use a single-table business view to access the data fields from the primary table, and use a business function or table I/O to access the data fields from secondary tables.

The best choice is not always obvious, but you can usually decide by examining the number of database I/O operations that will be performed. A joined business view that uses cross-data source joins causes slower performance.

If the secondary tables are "master" files, using a single-table business view is usually preferable because it makes better use of database caching. For example, suppose the primary table contains a company number and the related company name is stored in a secondary Company Master file. If, in practice, the same Company Master record is likely to be retrieved for several records in the primary table, then fetching the company name explicitly using a business function or table I/O is usually preferable.

EnterpriseOne is particularly optimized to fetch user defined codes. User defined code tables should never be included in a multiple-table business view.

Should I Restrict the Number of Fields in a Business View?

You might need to choose between using an existing business view that contains unused fields or creating a new business view with only the required fields. Although extraneous fields cause additional work both for the server and the workstation, they are usually not a significant performance concern. In cases for which tables will be updated, EnterpriseOne can automatically process all fields in a table, even if they do not exist in the business view.

Minimizing the number of fields in a data structure improves runtime performance better than minimizing the number of fields in a business view. You can create new business views that meet your specific needs and that do not affect performance.

Unions

Unions are the most expensive database operation of all the SQL SELECT statement types. Although we recommend that you avoid using them, unions might be appropriate for applications and processes (such as the process for posting journal entries) that use the same parts of tables but at different times.

See Also

- ❑ *The Post Process* in the *General Accounting Guide* for more information about the various tables that the system uses

Joined Business View Versus Table I/O

Use a joined business view if you are reading and updating two or more major tables. Use table I/O if the table being read or updated is merely an unintended result of another action.

Consider writing applications with the minimum number of grid columns required for the application's basic purpose. With minimal effort, you can add columns to the associated business view to supplement the basic features of the application. Therefore, adding columns to a business view is preferable to adding columns to a grid.

Data Structure Performance

This section discusses the performance ramifications of restricting the number of fields in a data structure and data structure objects.

Should I Restrict the Number of Fields in a Data Structure?

Performance measurements indicate that you should attempt to minimize the number of fields in a data structure, particularly when Configurable Network Computing (CNC) capabilities require that the data structure be passed between the server and the workstations.

Data Structure Objects

The following guidelines apply to data structure objects:

- Data structure header size is 237 bytes.
- Data structure item size is 72 bytes.
- Maximum number of data structure items to fit in 32K is $(32768-237)/72$ for 450 elements.

To provide space for literal values, limit your structure size to 350 elements or fewer. You should reconsider creating any data structure that exceeds 100 elements.

Data Selection and Sequencing

Use the following two system functions in the `Initialize Section` event to conditionally change the data selection for that section:

- Set User Selection
- Set Selection Append Flag

The `Initialize Section` event is normally used in the first level-one section of a report to conditionally select data based on values passed through the report data structure.

For example, Bill of Material Inquiry calls the Bill of Material report and passes the Parent Item, Parent Branch, Type of Bill, and Batch Quantity. In the `Initialize Section` event of the first level-one section of the Bill of Material report, the report values for the data structure are checked. If they are not equal to blank, the Set User Selection system function is used to add these selections.

Form Design

Use the following guidelines to increase performance across all form types.

- Limit the number of columns in the grid to the minimum required by the application.
- Keep the number of columns in the business view to the minimum required by the application.
- Limit the number of form controls, whether hidden or visible, to the minimum required by the application.
- Use event rule variables as work fields instead of hidden form controls.
- Disable data dictionary functions on form and grid controls that are not required, such as edits and default values, whether hidden or visible.
- Limit the amount of input and output performed for each grid row to the minimum required for the application. For example, avoid associated descriptions wherever possible.
- Use the Stop Processing system function whenever feasible to skip the processing of unnecessary event rules.
- Consider the design for the most efficient method of temporary data storage available at the time, such as cache versus either linked list or work files.

Find/Browse

Adherence to the following standards results in increased performance on Find/Browse forms.

- Do not use QBE assignments.
- The sort order on the grid should match both an index defined in EnterpriseOne and a logical defined on the iSeries, either partially or completely. The logical file and index must contain

at least all fields that are in the grid sort, and the fields selected for the grid sort must be in the same sequence as the logical/index fields. Additional fields might exist in the index or logical that are not included in the grid sort. For example, in a partial match, the grid sort can be KIT, MMCU; and the logical and index can include KIT, MMCU, TBM, BQTY.

Header Detail and Headerless Detail

Adherence to the following standards results in increased performance on Header Detail and Headerless Detail forms.

- The grid sort should match an index of the table in the business view. The index should exist both in the iSeries and the EnterpriseOne tables.
- The sort order on the grid should match both an index defined in EnterpriseOne and a logical file defined on the iSeries, either partially or completely. The logical and index must contain at least all fields that are in the grid sort, and the fields selected for the grid sort must be in the same sequence as the logical/index fields. Additional fields might exist in the index or logical that are not included in the grid sort. For example, in a partial match, the grid sort can be KIT, MMCU; and the logical and index can include KIT, MMCU, TBM, BQTY.

Batch Application Performance

This section discusses the performance ramifications of setting up level breaks. It also discusses the causes of slow performance.

Slow Performance

If the server side performance for your batch process is exceptionally slow and you cannot determine the cause, ask your system administrator to verify whether the `RequestedService` setting in the `jde.ini` file is set appropriately.

Event Rules Performance

When you create logic, if a row did not change then skip it. Do not check every row for changes.

Put master business function end docs on the `Post Button Clicked` event.

On asynchronous processing, put the `Post Button Clicked` event only on OK and Cancel. This ensures that your form does not terminate while the business function is still running. If the form terminates while the business function is still running, the business function cannot return values or handle errors.

Use system functions for repeated processes.

You can also use system functions to get to information that you cannot access, for example to write grid lines.

`FetchKeyed` is made up of `Clear Selection`, `Select Keyed`, and `Fetch`. Avoid using `FetchKeyed` unless the keys are changing.

If keys are not changing use `Select`, and then in a loop use `Fetch Next` for better performance, only if you do need to read records sequentially.

You should hide and show fields on the `Dialog is Initialized` event. You should put logic on the `Post Dialog is Initialized` event.

Always code `If` statements for the usual.

Use explicit comparisons instead of implied. It takes more code, but it runs faster.

You should make sure that the lines in a loop really need to be in there. If something does not really need to be done each time, then do not put it in a loop.

Reduce the number of returns inside code so you have only one exit point.

Use named variables to store temporary values. Do not use hidden controls or grid columns to store temporary values.

Using `jdeCache` Instead of Work Tables

In the past, developers used temporary local tables to contain working arrays. This technique is now obsolete. The `jdeCache` routines provide similar capabilities but with improved performance.

When multiple business functions access the same cache, locate the related functions in the same source member. For example, assume that an `EditGridLine` function adds records to a cache and a corresponding `EndDocument` function retrieves the cached records and inserts them into the database. If the business functions are placed in different source members, a change in the Configurable Network Computing™ configuration might cause the application to fail.

Choosing JDB API Calls in a Business Function or Table I/O Commands in Event Rule Code

When deciding whether to use JDB API calls in a business function or table I/O commands in event rule code to manage the database, let context guide your choice. Neither choice affects performance more than the other. For example, if event rule code needs to access the database, use event rule table I/O instead of writing and calling a business function.

Using `JDBFetchNext` versus Multiple `JDBFetch` Statements

To improve performance when multiple related database records must be read from the database, we recommend that you select the records at one time and use `JDBFetchNext` to loop through the qualifying records. The alternative, multiple calls to `JDBFetch`, is usually considerably slower.

Business Function Performance

You can have data dictionary edits in business functions.

If you have validation in your code and this validation is also part of a master business function, the tool will know that you have already validated and will return and say it is already done, instead of performing the validation twice.

If you go into a grid and do a custom select it is a big performance hit. You might want to restrict people from resequencing the grid. If there is a business need to resequence a grid, make sure there is an index over that table that matches the sort sequence.

Use business function caching to retain pieces in memory.

Do not use linked lists; use caching instead.

Create a structure and pass a pointer instead of adding items.

Verify that variables are used only if needed.

Do not use static variables. You should probably use a single record cache instead. An instance where you might want a static variable would be if you were getting parts of something repeatedly and totalling them.

Because `jdeAlloc` actually allocates space, you should not generally use it. You can, however, use `jdeAlloc` if you want to keep a storage area for multiple calls.

For example, suppose you have a function that is split into server and workstation components and you have parent/child information. The client makes a request for what the parent/child looks like. The server needs to know where to go back to. You can use `jdeAlloc` to keep the information in sequence.

`jdeCallObject` - When functions are built there is one `.c` in source. If you have more functions in here, then all functions run on the server if the main one does. You may, however, want some functions to run on the workstation and some on the server. Make sure that if you use more than one function that they are all completely dependent and you want them to run on the same place. `jdeCallObject` goes across DLLs and platforms.

Memory Allocation

Allocated memory that is not freed when it is no longer needed results in a memory leak. This can cause performance to degrade continuously as the application runs. You should be aware that `jdeCache` allocates memory. A cache that is created in a `BeginDoc` master business function must be destroyed in the corresponding `EndDoc` function to avoid memory leaks.

One frequent source of unexpected memory leaks comes from failing to free allocated memory when processing errors or other conditions. Failing to free allocated memory might cause the system to traverse an alternate execution path in a business function.

Review the `jdedebug.log` to identify possible memory leaks in business functions. You can also use a third-party tool for detecting memory leaks.

Balance Table Opens and Closes

Ensure that each `jdbOpen` is matched with a corresponding `jdbClose` within the same business function for all possible execution paths. Serious performance problems can arise from business functions within which table opens do not share a one-to-one correspondence with table closes.

Error Messaging Performance

You can classify error messages in the data dictionary as errors or warnings. An error appears as a red stop sign and a warning appears as a yellow yield sign. Validation is automatically performed using the data dictionary. You can create error messages that are validated by event rules. Ensure that you do the following:

- Use event rules to check in these error messages manually.
- Specify the field and error message number that the error is from.
- Code an If statement in event rules to prevent the system from redisplaying an error that is already displayed.

Note

If the error message exists in the data dictionary but not in event rules, the error displays every time.

Transaction Processing Performance

When you design applications that use transaction processing, you should maintain as narrow a scope as possible and you should start your transaction at a point that allows for the shortest possible time between the start of the transaction and the commitment or rollback. When you update a record that is part of a transaction, it is locked until the transaction is committed. If any part of a transaction fails, the entire transaction rolls back.

PeopleSoft EnterpriseOne Modification Rules

Because the PeopleSoft EnterpriseOne Tools are comprehensive and flexible, you can customize certain aspects of business solutions and applications without making custom modifications. PeopleSoft refers to this concept as "modless mods," which are modifications that you can perform easily without the help of a developer. You can perform modless mods on the following:

- User overrides
- User defined codes
- Menu revisions
- All text
- Processing option values
- Data dictionary attributes
- Workflow processes

The flexibility of modless mods improves efficiency and provides distinct advantages, such as the ability to do the following:

- Export grid records to other applications.
- Resequence a grid based on a different column.
- Change grid fonts and colors.
- Control major functionality using processing options.

However, even with the full commitment of PeopleSoft to making the tools as flexible and robust as possible, if it becomes necessary to modify EnterpriseOne, rules and standards exist to ensure that software modifications perform like modless mods for a seamless and predictable upgrade to the next release level.

You should prepare for the upgrade before making any custom modifications to ensure a smooth upgrade. If you plan modifications properly, you will have minimal work to do following an upgrade. Preparation and planning can result in the least amount of disruption to your business and reduced costs because upgrade time is shortened.

EnterpriseOne tracks all custom modifications as you check them into the server. Using this feature, you can run the Object Librarian Modifications Report (R9840D) before a merge to review a list of the changed objects.

EnterpriseOne consists of control tables (such as menus, user defined codes, versions, and the data dictionary) and transaction tables (such as Address Book Master (F0101) and Sales Order Header File (F4201)). The transaction files contain your business data. PeopleSoft ships the control tables with data that you can modify.

Both types of tables undergo an automatic merge process during an upgrade; control tables are merged with new EnterpriseOne data, while transaction files are converted to the new specifications while maintaining your existing data. For the object specification merges (such as business views, tables, data structures, processing options, event rules, and applications), processes are in place that merge or overlay the specifications, depending on the rules defined in the next section.

What an Upgrade Preserves and Replaces

If your business needs require custom software modifications, observe the following general terms for PeopleSoft EnterpriseOne modifications to ensure a smooth and predictable upgrade. These terms describe which of your modifications the upgrade process preserves and which modifications it replaces.

- "Preserve" means that, during an upgrade, the software automatically merges your modifications with the new applications shipped with the upgrade, and you do not lose your modifications. If your specifications conflict with system specifications, the upgrade process uses yours. When no direct conflict exists between the two, the upgrade process merges the two specifications.
- "Replace" means that the upgrade does *not* merge those types of modifications and, therefore, the system replaces your modifications. You will need to perform them again after the upgrade process completes.

You can run the Object Librarian Modifications Report (R9840D) before the upgrade process to identify objects that you modified.

General Rules for Modification

The following general modification rules apply to all PeopleSoft EnterpriseOne objects:

- When adding new objects, use system codes 55 through 59. EnterpriseOne uses reserved system codes that enable it to categorize applications and vertical groups. When you use system codes 55 through 59 for your custom modifications, EnterpriseOne does not overlay your modifications with EnterpriseOne applications.
- Do not create custom or new version names that begin with ZJDE or XJDE. These prefixes are reserved for standard version templates that PeopleSoft provides for you to copy for the purpose of creating new templates or versions. If your modifications use these prefixes, the system does not preserve your custom versions in case of a naming conflict.
- For upgrades, you should build a package from the last modified central objects set and perform backups of your development server, central objects, and Object Librarian data sources so that you can access those specifications for comparison or for troubleshooting purposes.

Interactive Applications

Do not delete controls, grid columns, or hyper-items on existing interactive applications. Instead, hide or disable them. EnterpriseOne might use these items for calculations or as variables, so deleting them might disable major functionality.

What an Upgrade Replaces

Custom forms in existing EnterpriseOne applications are replaced during an upgrade. Instead of placing custom forms on existing EnterpriseOne applications, you should create a custom application using system codes 55 through 59, and then place the custom forms on that custom application. You can then add to existing applications form exits and row exits that call your custom forms within your

custom applications. From a performance standpoint, the system recognizes no difference whether an external application is called from a row exit or a form within the application.

In addition to custom forms, the following interactive application elements are replaced during an upgrade:

Object	Comments
New applications	You can create an application or copy an existing application using the Application Design Aid's "Copy" feature. The Copy feature allows you to copy all of the application specifications, including event rules. Caution If you use the Copy feature to copy an existing application to which you add some modifications, during an upgrade your new application <i>will not</i> reflect changes that PeopleSoft might have made to the original application on which yours is based.
New hyper-items added to existing forms	
New controls added to existing forms	
New grid columns added to existing forms	
Style changes	Style changes include fonts and colors. New controls have standard base definitions. If you revise the style, you need to do the same for any new controls added to an application.
Code-generator overrides	
Data dictionary overrides	
Location and size changes	If PeopleSoft places a new control in the new release of the software in the same location that you have placed a custom control, the controls appear on top of each other. This does not affect the event rules or the functionality of the application. After the upgrade, you can use Application Design Aid to rearrange the controls.
Sequence changes for tabs or columns	The upgrade process adds new PeopleSoft EnterpriseOne controls to the end of your custom tab sequence. You can review the tab sequence after the upgrade.

What an Upgrade Preserves

The following interactive application elements are preserved during an upgrade:

Object	Comments
New applications	You can create a new application or copy an existing application by using the Copy feature within Application Design Aid. This application allows you to copy all of the application specifications, including event rules. If you use the Copy feature to copy an existing application for some modifications, during an upgrade your new application <i>does not</i> receive any changes made to the original application upon which your copy was based.
New hyper-items added to existing forms	
New controls added to existing forms	
New grid columns added to existing forms	
Style changes	Style changes include fonts and colors. New controls have standard base definitions. If you revise the style, you need to do the same for any new controls added to an application.
Code-generator overrides	
Data dictionary overrides	
Location and size changes	If EnterpriseOne places a new control in the new release of the software in the same place that you have placed a custom control, the controls appear on top of each other. This does not affect the event rules or the functionality of the application. After the upgrade, you can use Application Design Aid to rearrange the controls.
Sequence changes for tabs or columns	The upgrade process adds new EnterpriseOne controls to the end of your custom tab sequence. You can review the tab sequence after an upgrade.

Reports

The following rule applies to Report Design Aid specifications:

Do not delete objects on existing PeopleSoft EnterpriseOne software reports. Instead, hide them. The system might use these for calculations or as variables, and deleting them might disable major functionality.

What an Upgrade Preserves

The following report elements are preserved during an upgrade:

Object	Comments
New reports	You can create a report either from scratch, or by copying an existing report by using the "Copy" feature within Report Design Aid. This feature enables you to copy all of the report specifications, including event rules. If you use the Copy feature to copy an existing report for some modifications, during an upgrade your new report <i>does not</i> receive any updates that PeopleSoft EnterpriseOne made to the original report on which your copy was based.
New constants added to existing reports	
New alpha variables added to existing reports	
New numeric variables added to existing reports	
New data variables added to existing reports	
New runtime variables added to existing reports	
New database variables added to existing reports	
New data dictionary variables added to existing reports	
Style changes	Style changes include fonts and colors. New PeopleSoft EnterpriseOne controls have standard base definitions. If you revise the style, you need to do the same for any new controls added to a report.
Location and size changes for objects	If PeopleSoft places a new object, such as a control, in the new release of the software in the same place that you have placed a custom object, the objects appear next to each other. This placement does not affect the event rules or the functionality of the report in any way. After the upgrade, you can use Report Design Aid to rearrange objects.
Data dictionary overrides	

What an Upgrade Replaces

During an upgrade, the system replaces custom sections in existing PeopleSoft EnterpriseOne software reports. Instead of adding custom sections to existing reports, use Report Interconnect and connect to a new custom report that uses system codes 55 through 59. System performance is not affected if a report is called through report interconnections.

Application Text Changes

What an Upgrade Preserves

The following application text elements are preserved during an upgrade.

- Overrides created in Application Design Aid
- Overrides created in Report Design Aid
- Overrides created in Interactive Vocabulary Override
- Overrides created in Batch Vocabulary Override

Table Specifications

An upgrade merges your table specifications from one release level to the next. The following table shows which table specification elements are preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
New tables	X		
Custom indexes to PeopleSoft EnterpriseOne tables	X		
Columns added to or removed from existing EnterpriseOne tables		X	These modifications include changing field length, field type, and decimal position. Instead of adding a new column to an existing EnterpriseOne table, use a tag file with system codes 55 through 59.

Note

For custom tag files, be aware of data item changes in the EnterpriseOne data dictionary. From one release to the next, PeopleSoft might change certain data item attributes, such as data item size, which can affect data integrity and how data is stored in the database. For this reason, you might need to use the Table Conversion tool to convert the tag file data to the new release level. For base EnterpriseOne files, the upgrade process performs the data dictionary changes by upgrading the EnterpriseOne database to the new release level. The upgrade process preserves custom indices over the custom tag files.

Control Tables

Control tables contain user defined codes (UDCs), menus, and data dictionary items. An upgrade merges your control tables from one release level to the next using the Change Table process, which uses your control tables, *not* PeopleSoft EnterpriseOne tables, as the basis on which the data is merged.

The following table shows which control table elements are preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
Data dictionary custom changes	X		Includes changes to row, column, and glossary text. The upgrade process uses your data dictionary as the base. If your values conflict with EnterpriseOne data items, your values override the EnterpriseOne values. Create new data items using system codes 55 through 59.
UDCs	X		The upgrade process merges new hard-coded EnterpriseOne values. (Values owned by PeopleSoft are system 90 and above, and H90 and above.) The process also reports PeopleSoft hard-coded values that conflict with your custom values.
Menus	X		If your custom menus conflict with EnterpriseOne base menus, your custom changes override the base menus.
Columns added or removed from existing EnterpriseOne control tables		X	

Business Views

Do not remove columns from existing business views. Modifications to business views that applications use can cause unpredictable results when you run the applications. If you need to hide columns, do so at the application design level using either Application Design Aid or Report Design Aid. Performance that might be gained by deleting a few columns from a business view is negligible.

The following table shows which business view elements are preserved or replaced during an upgrade:

Object	Preserved	Replaced
New custom business views	X	
New columns, joins, or indexes added to existing PeopleSoft EnterpriseOne software business views	X	
Columns removed from EnterpriseOne business views		X

Data Structures

The following table shows which data structure elements are preserved or replaced during an upgrade:

Object	Preserved	Replaced
Custom forms data structures	X	
Custom processing-options data structures	X	
Custom reports data structures	X	
Custom business-function data structures	X	
Custom generic-text data structures	X	
Modifications to existing PeopleSoft EnterpriseOne form data structures		X
Modifications to existing EnterpriseOne processing-option data structures		X
Modifications to existing EnterpriseOne report data structures		X
Modifications to existing EnterpriseOne business-function data structures		X
Modifications to existing EnterpriseOne generic-text data structures		X

To move custom modifications made to EnterpriseOne data structures forward to the next release level, run the Object Librarian Modifications Report (R9840D) to list all of the modified data structures. Use the resulting report as a guide for manually refitting data structure changes.

Event Rules

Before you begin the upgrade process, PeopleSoft recommends that you complete the following procedures:

- Run the Object Librarian Modifications Report (R9840D) to identify modified applications.
- Print the event rules for modified applications so that you can review the logic for the event when you restore custom event rules.

The following table shows which event rules elements are preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
Custom event rules for custom applications, reports, and tables	X		
Custom event rules for custom business functions	X		
Custom event rules on a new custom control	X		
Events for PeopleSoft EnterpriseOne applications, reports, and tables that do not have any EnterpriseOne event rules attached to the same event	X		
Events for EnterpriseOne business functions that do not have any EnterpriseOne event rules attached to the same event	X		
Events for EnterpriseOne applications, reports, and tables that have existing event rules attached to the same event		X	An upgrade disables and appends your custom event rules to the end of the event rules. The system generates the Object Librarian Modifications Report (R9840D) to notify you of any event rules that the upgrade process disabled.
Events for EnterpriseOne business functions that have event rules attached to the same event		X	An upgrade disables and appends your custom event rules to the end of the event rules. The system generates the Object Librarian Modifications Report (R9840D) to notify you of any event rules that the upgrade process disabled.

To restore your custom event rules to EnterpriseOne objects, drag the event rules back to the correct location in the event and enable them.

Versions

Before an upgrade, verify that new custom versions are named such that they do not begin with XJDE or ZJDE.

The following table shows which versions elements are preserved or replaced during an upgrade:

Object	Preserved	Replaced
Non-JDE versions	X	
Version specifications	X	
Processing option data	X	
All ZJDE and XJDE version specifications		X
All processing option data for XJDE versions		X

In addition to the information in the preceding table, the system copies processing option data but does not convert it for non-JDE versions using JDE processing option templates. The system generates a warning at runtime. Some data might be lost. Additionally, event rule modifications for custom versions on JDE templates are not reconciled with the JDE parent template.

Business Functions

For any new custom business functions, create a new (custom) parent DLL to store your custom modifications.

To move your custom changes forward to the next release level, run the Object Librarian Modifications Report (R9840D) to list all of the modified business functions, and then use this report as a guide for manually refitting the business function changes.

To prevent unpredictable results, always use the standard API (jdeCallObject) to call other business functions from within a business function.

To determine modifications to the source of existing base PeopleSoft EnterpriseOne business functions, use a third-party source-compare tool, such as Microsoft WinDiff. To determine modifications to APIs within business functions, see the EnterpriseOne online help feature for the most current APIs.

The following table shows which business function elements are preserved or replaced during an upgrade:

Object	Preserved	Replaced	Comments
New custom business function objects	X		
Modifications made to existing EnterpriseOne business function objects		X	NER business functions can be modified.

Date Reference Scan

You can run date scan programs to search for date references in event rules or business functions. PeopleSoft provides two date-reference scanning programs; one scans business functions, and the other scans event rules. When you scan business functions for references to dates, the scanning program also scans for system date references.

Business Function Date Reference Scan

You can run a business function date reference scan to check business functions for date references. This scan also reports references to system date functions. You can then review the reports that the system creates during this process. You can use the following PeopleSoft EnterpriseOne date reference scans:

- Business function date references scan
- Event rule date references scan

► To run the Business Function Date Reference Scan Report (R9000085)

1. On Object Management Workbench, check out the following objects to transfer the required specifications to your workstation:
 - B9000085 - Business Function Date Reference Scan Report
 - D9000085 - Data Structure for Date Reference Scan
 - R9000085 - Business Function Date Reference Scan Report
2. Choose B9000085 and click the Design button.
3. Click the Design Tools tab, and then click Build Business Function.
4. Choose the version that you want to run and click Select.
5. On Data Selection and Sequencing, choose the options that you want to use and click Submit.
6. Check your PDF report and the text log file in the default print directory, which is usually the B9\PrintQueue directory.

When you run the Business Function Date Reference Scan Report (R9000085), the system generates two versions of the report. One version indicates the number of date references that the scan program detected. The other version provides detailed information about the date references.

Event Rule Date Reference Scan

You can scan event rules to verify event rules for date references. You can then review the reports that the system generates during this process.

The process to scan event rules is the same as the process to scan business functions. Use the following objects:

- B9000085 - Business Function Date Reference Scan Report
- D9000085 - Data Structure for Date Reference Scan
- R9000085 - Business Function Date Reference Scan Report

When you run the Business Function Date Reference Scan Report (R9000085), the system creates two reports. One report indicates the number of date references that the scan program detected. The other report provides detailed information about the date references.

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 other 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 serviços.
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 reusable 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.I Inheritance. Encapsulation.
object-oriented technology (OOT)	Brings software development past procedural programming into a world of re-usable 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.
repassse	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

- Access32, 316
- Action messages, 278
- Active messages, 263
- Add a processing option template with translated text, 243
- Add button
 - processing, 87
- Adding
 - business function
 - object codes, 177
 - business functions
 - codes, 177
 - related functions, 178
 - related tables and functions, 177
 - comment lines to event rules, 95
- Adding a business view, 42
 - joined views, 44
 - Naming standards, 43
- Adding glossary text for languages, 28
- Adding table
 - indices, 33
- Adding tips of the day to an object, 261
- Additional features
 - currency, 255
 - processing options, 238
 - record locking, 253
 - transaction processing, 244
- Advanced functions
 - expression manager, 124
 - date functions, 124
 - general functions, 125
 - text functions, 126
 - trig functions, 125
- Advantages
 - currency advantages, 256
- Alias
 - used in an RPG program, 19
- Alias for an external data dictionary item, 19
- All grid recs deleted from DB
 - runtime processing, 91
- APIs
 - JDECACHE cursors
 - cursor-advancing APIs, 229
 - non-cursor-advancing APIs, 229
 - JDECACHE manipulation, 218
 - jdeCacheFetchPosition, 230
 - jdeCacheFetchPositionByRef, 230
- Application
 - currency, 257
 - debugging an application, 300
 - inspecting or modifying a variable, 301
 - event rules, 67
 - JDECACHE, 216
- Application event rules, 12
- Application performance tips, 315
- Application Programming Interfaces, 168
- Application specific parameters
 - begin document, 194
 - end document, 198
- Application text modification rules, 331
- Applications, 6
 - building an application, 7
 - change row and column text, 30
 - messaging, 262
- Assign
 - value, 123
- Assignment
 - creating
 - expression, 123
- Assignment display
 - expression manager, 124
- Asynchronous
 - business functions, 140
- Asynchronous events
 - how threaded events work, 138
- Asynchronous processing, 138
- Attaching
 - business function to an event, 117
 - system function to an event, 103
- Attaching a data structure template to a message, 272
- Attaching a default value trigger, 22
- Attaching a display rule trigger, 24
- Attaching a next number trigger, 25
 - next numbers, 25
- Attaching a processing options template, 243
- Attaching a smart field trigger, 26
- Attaching a user defined codes trigger, 25
- Attaching a visual assist trigger, 22
- Attaching an edit rule trigger, 23
- Attaching an interactive message data item, 273
- Attaching default triggers, 21
- Automatic error settings, 265
- Automatic Line Numbering, 103

- Available business functions
 - business function builder form, 180
- Available information
 - expression manager, 124
- Available objects
 - processing, 70
- Available objects and runtime data structures, 68
- Available objects tab, 53
- Available operations
 - Table I/O, 128

B

- Balance table opens and closes
 - business function performance, 324
- Batch application event rules variables, 101
- Batch application performance, 322
 - slow performance, 322
- Batch applications
 - debugging business functions, 302
 - searching for batch applications, 60
- Batch error messages, 276
 - action messages, 278
 - creating a level-break message, 279
 - first-level messages, 277
 - level break messages, 276
 - level-break messages, 276
 - second-level messages, 277
 - text substitution error messages, 277
 - third-level messages, 277
 - work center API, 278
- Batch error messaging, 276
- BC assigned database values
 - runtime processing, 77
- Begin DLL section
 - build output, 183
- Begin document
 - application specific parameters, 194
 - common parameters, 193, 195
 - function name, 192, 194
 - hook up tips, 192
 - naming
 - transaction master business function modules, 192
 - special logic/processing required, 192, 195
 - typical users and hook up, 195
 - what does it do, 192
- Breakpoint manager, 299
- Breakpoints
 - setting breakpoints, 301, 304
- BrowsER
 - expanded node, 145
 - filter ER records, 145

- options
 - working with, 145
- search, 145
- show object ID, 145
- working with, 144
- Build all
 - performing, 185
- Build options
 - setting, 181
- Build output
 - begin DLL section, 183
 - business function builder form, 180
 - compile section, 184
 - link section, 184
 - makefile section, 183
 - reading, 183
 - rebase section, 184
 - summary section, 185
- Build triggers option, 256
- Building
 - single business function, 181
 - transaction master business functions, 202
- Building an application, 7
- Business function
 - cache business function description, 234
 - cache business function source description, 233
 - cache business function source name, 233
 - creating
 - documentation, 209
 - documentation
 - generating, 211
 - template, 210
 - event
 - attaching, 117
 - event rules, 67
 - generating
 - documentation, 211
 - level-break messages, 281
 - modification rules, 335
 - processing failover, 214
- Business function - values to pass
 - documentation
 - viewing, 213
- Business function builder
 - resolving errors, 189
 - working with, 179
- Business Function Builder
 - DLLs, 150
- Business function builder form
 - available business functions, 180
 - Build output, 180

- functions list, 180
- understanding, 180
- Business function data reference scan, 336
- Business function data structure, 54
- Business function data structures, 52
 - interconnection data structures, 52
- Business function design, 11
- Business Function Design form, 179
- Business function document viewer
 - documentation
 - viewing, 214
- Business function event rules, 11
 - creating, 154
 - creating or editing event rules, 154
- Business Function Object Codes
 - adding, 177
- Business function objects
 - linking, 181
- Business function performance, 323
 - balance table opens and closes, 324
 - memory allocation, 324
- Business function search
 - documentation
 - viewing, 213
- Business function source files
 - structure, 163
- Business functions
 - adding
 - codes, 177
 - object codes, 177
 - related functions, 178
 - related tables and functions, 177
 - asynchronous, 140
 - building a single, 181
 - changing
 - parent DLL, 178
 - components, 150
 - creating a level-break business function, 282
 - debugging, 178
 - debugging by using Microsoft Visual C++, 301
 - extending a transaction boundary using
 - business functions, 251
 - J.D. Edwards tools, 149
 - pessimistic record locking, 255
 - searching for business functions, 61
 - transaction processing, 248
 - transaction processing in remote business
 - functions, 248
 - using the same cache, 225
- Business Functions
 - building all, 185

- Business view
 - adding a business view, 42
 - joined views, 44
 - Naming standards, 43
 - choosing a table, 45
 - choosing data items, 46
 - creating a table join, 50
 - creating a table union, 51
 - searching for business views, 61
 - using select distinct, 47
- Business view design, 9, 40, 44
 - indices, 42
 - select distinct, 42
 - table join, 40
 - table union, 41
- Business view performance, 319
 - performance tips
 - joined business view versus table I/O, 320
 - restrict the number of fields, 320
 - single-table or multiple-table, 319
 - table I/O versus joined business view, 320
 - unions, 320
- Business views
 - modification rules, 332
- Button clicked
 - runtime processing, 86

C

- Cache
 - structure, 199
 - detail, 201
 - header, 200
 - using the same cache in multiple business
 - functions or forms, 225
- Cache action code standards, 235
 - CACHE_ADD, 236
 - CACHE_ADD_UPDATE, 236
 - CACHE_CLOSE_CURSOR, 237
 - CACHE_DELETE, 236
 - CACHE_GET, 235
 - CACHE_GET_NEXT, 237
 - CACHE_TERMINATE, 237
 - CACHE_TERMINATE_ALL, 237
 - CACHE_UPDATE, 236
- Cache business function description, 234
- Cache business function source name, 233
- Cache data structure
 - group cache function, 235
 - individual cache function, 235
- Cache name, 235
- Cache programming standards, 234
 - cache action code standards, 235

- cache name, 235
- data structure standard data items, 235
- defining the cache data structure, 235
- general standards, 234
- terminating versus clearing cache, 234
- CACHE_ADD, 236
- CACHE_ADD_UPDATE, 236
- CACHE_CLOSE_CURSOR, 237
- CACHE_DELETE, 236
- CACHE_GET, 235
- CACHE_GET_NEXT, 237
- CACHE_TERMINATE, 237
- CACHE_TERMINATE_ALL, 237
- CACHE_UPDATE, 236
- Caching, 216
 - initializing the cache, 223
 - JDECACHE, 216, 219
 - performance considerations, 217
 - JDECACHE API set, 217
 - JDECACHE APIs, 216
 - JDECACHE cursors, 226
 - opening JDECACHE cursors, 226
 - tables, 216
- Calling the processing work center API, 288
- Calling the work center initialization API, 287
- cAllowUserIdToChange parameter, 289
- Cancel document (optional)
 - common parameters, 199
 - function name, 199
 - naming
 - transaction master business function modules, 199
 - special logic/processing required, 199
 - what does it do, 199
- Central object data dictionary
 - storing data dictionary and data dictionary items, 15
- Changing
 - business function
 - parent DLL, 178
- Choosing a data item for a business view, 46
- Choosing a table for a business view, 45
- Clear cache
 - function name, 198
 - naming
 - transaction master business function modules, 198
 - special logic/processing required, 198
 - what does it do, 198
- Closing a cursor, 231
- Code
 - interpretive and compiled code, 296
- Coexistence
 - data item used in RPG program, 19
 - indices and logicals, 33
 - issues addressed in this guide, 18
- Column
 - exited and changed - asynch, 140
- Column properties, 39
- Comment, 242
 - entering a comment for a processing option, 242
- Comment lines
 - event rules
 - adding, 95
- Commit, 245
 - two-phase commit (manual commit mode), 245
- Commits and rollbacks, 245
 - commit, 245
 - rollback, 245
- Common parameters
 - begin document, 193, 195
 - cancel document (optional), 199
 - edit document, 197
 - end document, 198
- Compile section
 - build output, 184
- Components
 - business functions, 150
- Connecting
 - database, 173
- Control
 - exited and changed - asynch, 140
 - exited processing, 71
- Control tables modification rules, 332
- Controls, 65
 - understanding, 65
- Copying
 - event rules, 94
- Copying tables, 36
- Create multi-level error messaging, 269
- Creating
 - business function documentation, 209
 - business function event rules, 154
 - data structure documentation, 210
 - event rule variables, 101
 - expression for an assignment, 123
 - if and while statements, 98
 - master business function
 - processing options, 190
 - modal form interconnect, 119
 - modeless form interconnect, 120
 - parameter documentation, 211

- table event rules, 135
- Table I/O event rule, 129
- Creating a business function data structure, 54
- Creating a currency conversion trigger, 259
- Creating a data dictionary item for a level-break message, 279
- Creating a data structure for the data dictionary item, 280
- Creating a level-break business function, 282
- Creating a level-break message, 279
- Creating a level-break message business function structure, 281
- Creating a media object data structure, 55
- Creating a processing options data structure, 56, 240
- Creating a simple error message, 271
- Creating a table join, 50
- Creating a table union, 51
- Creating a text substitution error message, 271
- Creating and saving
 - event rules, 92
- Creating data structures, 54
- Creating or editing
 - event rules for a business function event rules, 154
- Cross reference facility, 59
 - rebuilding cross reference information, 64
 - searching for batch applications, 60
 - searching for business functions, 61
 - searching for business views, 61
 - searching for data items, 59
 - searching for data structures, 62
 - searching for event rules, 63
 - searching for forms, 62
 - searching for interactive applications, 60
 - searching for objects, 59
 - searching for tables, 62
 - viewing field relationships, 63
- Cross reference information
 - rebuilding cross reference information, 64
- Currency, 255, 256
 - advantages, 256
 - application, 257
 - build trigger options, 256
 - creating a currency conversion trigger, 259
 - currency implementation, 255
 - setting up currency conversion, 258
 - showing currency-sensitive controls, 258
- Currency conversion
 - setting up currency conversion, 258
- Currency conversion trigger, 259
- Currency implementation, 256

- Currency process
 - table event rules, 257
- Cursor-advancing APIs, 229
- Cursors
 - JDECACHE cursors, 226
 - opening JDECACHE cursors, 226
- Cutting
 - event rules, 94

D

- Data dictionary, 9, 13
 - alternate language terms, 28
 - data structure, 280
 - default triggers, 16
 - error messages, 16
 - glossary items, 16
 - jargon, 28
 - level-break message, 279
 - naming data items, 15
 - runtime, 14
 - storing data dictionary and data dictionary items, 15
 - using the data dictionary, 17
- Data dictionary and data dictionary items
 - storing Data Dictionary and Data Dictionary items, 15
- Data dictionary item naming conventions
 - data item alias, 19
 - data item name for an external data dictionary item, 18
- Data dictionary items
 - naming conventions, 19
- Data dictionary naming conventions, 18
- Data dictionary performance
 - overrides, 315
 - triggers, 315
 - validation, 315
- Data interdependence, 246
- Data item, 243
 - attaching and interactive message data item, 273
 - attaching default triggers, 21
 - change row and column text for all applications, 30
 - choosing a data item for a business view, 46
 - general information, 20
 - naming a data item, 20
 - naming data items, 15
 - searching for data items, 59
 - selecting a data item for a processing option, 243
 - updating the glossary, 28

- Data Item
 - alias, 19
- Data item alias, 19
 - alias for an external data dictionary item, 19
- Data item name, 18
- Data item name for an external data dictionary item, 18
- Data item specifications
 - defining general information, 20
- Data items
 - choosing data items for the table, 34
- Data Items form, 30
- Data reference scan
 - business function data reference scan, 336
 - event rule data reference scan, 337
- Data Reference Scan, 336
- Data selection and sequencing, 321
- Data structure
 - creating
 - documentation, 210
 - creating a processing options data structure, 240
 - documentation
 - template, 210
- Data structure objects
 - data structure performance
 - performance tips, 320
- Data structure performance, 320
 - performance tips
 - data structure objects, 320
 - restrict the number of fields, 320
- Data structure standard data items, 235
- Data structure template
 - attaching to a message, 272
- Data structures, 51
 - business function data structures, 52
 - create a media object data structure, 55
 - creating a business function data structure, 54
 - creating a media object data structure, 55
 - creating a processing options data structure, 56
 - creating data structures, 54
 - defining, 57
 - displaying type definitions, 54
 - interconnection data structures, 52
 - modifying form data structures, 52
 - available objects tab, 53
 - modifying report data structures, 53
 - searching for data structures, 62
 - system generated, 51
 - system generated data structures
 - form data structures, 51
 - report data structures, 51
 - type definition, 59
 - user generated data structures, 52
 - media object data structures, 52
 - processing options data structures, 52
- Data structures modification rules, 333
- Data type
 - JDEDATE, 170
 - MATH_NUMERIC, 169
 - PeopleSoft EnterpriseOne
 - specific, 168
- Database
 - connecting, 173
 - index limitations, 316
 - Access 32, 316
 - DB2 for OS/400, 317
 - key column violation, 318
 - Oracle, 317
 - specification file corruption, 318
 - SQLSERVER, 317
- Database APIs
 - using, 170
- Database communications steps
 - understanding, 173
- Database locking, 254
- Date functions
 - advanced functions
 - expression manager, 124
- DB2 for OS/400, 317
- Debug logs, 307
- Debug tracing, 309
 - system function tracing, 309
 - turning on debug tracing, 309
- Debugger tools
 - Microsoft Visual C ++, 291
- Debugging, 291
 - business functions, 178, 301
 - batch applications, 302
 - interactive applications, 301
 - debug tracing, 309
 - system function tracing, 309
 - turning on debug tracing, 309
 - debugging an application, 300
 - inspecting or modifying a variable, 301
 - debugging features, 291
 - debugging process, 291
 - debugging strategies, 306
 - application encountering errors, 306
 - debug logs, 307
 - function being called as expected, 307
 - output of the program incorrect, 307
 - program ending unexpectedly, 306

- source of the problem, 307
 - SQL log tracing, 308
- event rules debugger, 297
 - breakpoint manager, 299
 - event rules window, 297
 - object browse window, 297
 - search combo box, 299
 - variable selection and display window, 298
- features of Visual C++ debugger, 303
 - go command, 303
 - local window, 304
 - setting breakpoints, 304
 - step command, 303
 - step into command, 303
 - using watch, 304
- interpretive and compiled code, 296
- modify the Visual C++ debugging environment, 305
- setting breakpoints, 301
- tools, 291
- Visual C++ debugger, 303
- Debugging an application, 300
 - inspecting or modifying a variable, 301
- Debugging business functions attached to batch applications, 302
- Debugging business functions attached to interactive applications, 301
- Debugging business functions using Microsoft Visual C++, 301
- Debugging features, 291
- Debugging strategies, 306
 - application encountering errors, 306
 - debug logs, 307
 - function being called as expected, 307
 - output of the program incorrect, 307
 - program ending unexpectedly, 306
 - source of the problem, 307
 - SQL log tracing, 308
- Debugging tools
 - EnterpriseOne software Event Rules, 291
- Default triggers, 16
 - attaching default triggers, 21
 - default value trigger, 22
 - display rule trigger, 24
 - edit rule trigger, 23
 - next number trigger, 25
 - smart field trigger, 26
 - user defined codes trigger, 25
 - visual assist trigger, 22
- Default value trigger, 22
- Defining
 - literal values in event rule logic, 100
 - Defining an active message, 275
 - Defining general information, 20
 - updating data items for languages, 30
 - Defining indices, 34
 - Defining jargon, 29
 - Defining the cache data structure, 235
 - Defining the glossary text for a runtime message, 272
 - Defining transaction processing for a form, 249
 - Defining transaction processing for a report, 252
 - Delete button
 - processing, 89
 - Delete grid rec from DB after runtime processing, 91
 - Delete grid rec from DB before runtime processing, 91
 - Delete grid rec verity after runtime processing, 90
 - Delete grid rec verity before runtime processing, 90
 - Deleting records, 230
 - Design standards
 - EnterpriseOne, 118, 120
 - Detail
 - cache structure, 201
 - Developing Web applications, 311
 - Development cycle, 7
 - Development tools
 - Web applications, 311
 - Dialog
 - initialized, 73
 - Dictionary item naming conventions
 - data item name, 18
 - Display rule trigger, 24
 - Displaying type definitions, 54
 - Distributed business functions, 152
 - DLLs, 150
 - Do in while loop
 - runtime processing, 77
 - Document processing
 - transaction master business function implementing, 204
 - Documentation
 - business function
 - creating, 209
 - template, 210
 - business function - values to pass viewing, 213
 - business function document viewer viewing, 214
 - business function search viewing, 213

- data structure
 - creating, 210
 - template, 210
- generating
 - business function, 211
- parameter
 - creating, 211

E

- Edit document
 - common parameters, 197
 - hook up tips, 196
 - naming
 - transaction master business function modules, 196
 - special logic/processing required, 196
 - what does it do, 196
- Edit line
 - naming
 - transaction master business function modules, 194
 - what does it do, 194
- Edit rule trigger, 23
- Editing keypad
 - expression manager, 124
- Editing or creating
 - event rules
 - business function event rules, 154
 - event rules for a business function event rules, 154
- e-mail
 - workflow messages, 263
- Embedded
 - event rules, 67
- Embedded event rules, 11
 - application event rules, 12
 - table event rules, 12
- End document
 - application specific parameters, 198
 - common parameters, 198
 - function name, 197
 - hook up tips, 198
 - naming
 - transaction master business function modules, 197
 - special logic/processing required, 198
 - what does it do, 197
- Enterprise server
 - jde.ini
 - setting, 252
- EnterpriseOne
 - design standards, 118, 120

- EnterpriseOne naming standards
 - guidelines when adding a business view
 - external developer considerations, 43
- EnterpriseOne processing option naming standards, 241
- EnterpriseOne standards
 - processing option naming standards, 241
- EnterpriseOne storing objects, 8
- EnterpriseOne tools, 9
 - business view design, 9
 - development cycle, 7
 - Event Rules Design, 65
 - table design, 9
- Environment
 - customizing the Visual C++ debugging environment, 305
- Error handling, 264
 - error setting, 265
 - error settings
 - automatic error settings, 265
 - manual error settings, 265
 - event-driven model, 264
 - multilevel error messaging for C business functions, 268
 - resetting errors, 267
- Error messages, 16, 263, 270
 - attaching an interactive message data item, 273
 - batch error messages, 276
 - creating a level-break message, 279
 - creating a simple error message, 271
 - creating a text substitution error message, 271
 - locating an existing error message, 270
- Error messaging performance, 325
- Error output
 - reviewing, 189
- Error Setting, 265
- Error settings
 - automatic error settings, 265
 - manual error settings, 265
- Errors
 - resolving, 189
 - business function builder tools, 189
- Event
 - attaching a business function, 117
 - attaching a system function, 103
- Event flow
 - typical
 - Find/Browse form, 72
- Event rule
 - form flow, 72
 - manipulation, 70

- modification rules, 335
- Table I/O
 - creating, 129
- Event rule data reference scan, 337
- Event rule logic
 - if statements, 96
 - literal values
 - defining, 100
 - printing, 95
 - while statements, 97
- Event rule variables
 - batch application, 101
 - creating, 101
 - interactive, 100
- Event rules, 11
 - adding
 - comment lines, 95
 - application, 67
 - business function, 67
 - business function event rules, 11
 - creating or editing, 154
 - copying, 94
 - creating and saving, 92
 - currency
 - build triggers options, 256
 - currency processing, 257
 - cutting, 94
 - embedded, 67
 - embedded event rules, 11
 - paste, 94
 - searching for event rules, 63
 - table, 67
 - understanding, 66
 - validating, 96
- Event rules buttons
 - understanding, 67
- Event rules debugger, 297
 - debugging an application, 300
 - inspecting or modifying a variable, 301
 - main controls, 297
 - search combo box, 299
- Event Rules Design
 - EnterpriseOne tools, 65
 - Table I/O, 127
- Event rules performance, 322
 - performance tips
 - I/O commands in event rule code, 323
 - JDB API calls in a business function, 323
 - JDBFetch statements, 323
 - JDBFetchNext, 323
 - jdeCache instead of work tables, 323
- Event rules window, 297

- Event-driven model, 264
- Event-level
 - interactive event rule variables, 101
- Events, 66
 - understanding, 66
- Exited and changed - asynch
 - Column, 140
 - control, 140
 - row, 140
- Exited processing
 - control, 71
- Expanded node
 - BrowsER, 145
- Expression
 - creating for an assignment, 123
- Expression editing field
 - expression manager, 124
- Expression manager, 123
 - advanced functions, 124
 - date functions, 124
 - general functions, 125
 - text functions, 126
 - trig functions, 125
 - assignment display, 124
 - available information, 124
- Editing keypad, 124
 - expression editing field, 124
- Extending a transaction boundary, 250
- Extending a transaction boundary between
 - forms, 250
- Extending a transaction boundary using business
 - functions, 251
- Extending a transaction boundary using table
 - I/O, 251
- External Business Function
 - calling an API from, 174
- External data dictionary item, 18
- External data dictionary Item
 - alias for an external data dictionary item, 19
- External developer considerations
 - EnterpriseOne naming standards
 - guidelines when adding a business view, 43

F

- F0002 - Next Number table, 25
- F00165 - Media object storage, 55
- Field relationships
 - viewing field relationships, 63
- Filter ER Records
 - BrowsER, 145
- Find/Browse

- form design performance tips, 321
- Find/Browse form
 - typical event flow, 72
- First-level messages, 277
- Foreign currency, 255
- Form
 - defining transaction processing for a form, 249
- Form data structures, 51
- Form design, 10, 321
 - find/browse, 321
 - header detail and headerless detail, 322
- Form flow
 - event rule, 72
- Form processing, 66
- Format tables
 - performance impact, 208
- Formatted data, 39
- Form-level
 - interactive event rule variables, 100
- Forms
 - business function builder
 - understanding, 180
 - Business Function Design, 179
 - controls
 - understanding, 65
 - Data Items, 30
 - events
 - understanding, 66
 - extending a transaction boundary, 250
 - extending a transaction boundary between forms, 250
 - form processing, 66
 - searching for forms, 62
 - using the same cache, 225
- Function name
 - begin document, 192, 194
 - cancel document (optional), 199
 - clear cache, 198
 - end document, 197
- Functions
 - distributed business, 152
- Functions list
 - business function builder form, 180
- Fundamentals
 - business view design, 40
 - cross reference facility, 59
 - data dictionary, 13
 - data structures, 51
 - table design, 31

G

- General Functions
 - advanced functions
 - expression manager, 125
- General standards
 - cache APIs, 234
- Generating header files, 36
- Generating indexes, 36
- Generating tables, 35
- GH951 - Data Dictionary menu, 17
- Glossary
 - updating the glossary, 28
- Glossary items, 16
- Glossary text
 - defining for a runtime message, 272
- Go command, 303
- Grid
 - processing, 71
 - set lower limits, 71
- Grid rec is fetched
 - runtime processing, 78
- Group cache function, 235

H

- Handle request
 - event rule variable, 102
- Header
 - cache structure, 200
- Header detail and headerless detail
 - form design performance tips, 322
- Header file sections, 158
- HJDECURSOR, 227
- Hook up tips
 - begin document, 192
 - edit document, 196
- How JDECACHE partial key works, 233
- How table event rules work with currency processing, 257
- How the data dictionary is used at runtime, 14
- How to build an application, 7
- Hungarian notation
 - example of, 102

I

- I/O commands in event rule code
 - event rules performance
 - performance tips, 323
- If and while statements
 - creating, 98
- If statements
 - event rule logic, 96
- Implementing

- transaction master business functions, 203
 - document processing, 204
 - single record processing, 203
- Index
 - defined, 31
 - generate for tables, 36
 - using an index to access the cache, 224
- Index considerations, 316
- Index limitations for various databases, 316
 - Access 32, 316
 - DB2 for OS/400, 317
 - key column violation, 318
 - Oracle, 317
 - specification file corruption, 318
 - SQLSERVER, 317
- Indices, 33, 42
 - defining indices, 34
 - JDECACHE, 220
 - setting up indices, 220
- Indices and logicals
 - coexistence, 33
- Individual cache function, 235
- Information messages
 - glossary items, 16
- Information Messages, 263
- Information structure
 - standard parameters, 207
- Initialized
 - dialog, 73
 - post dialog, 74
 - pre dialog, 73
- Initializing the cache, 223
- Inspecting or modifying a variable, 301
- Interactive applications
 - debugging business functions, 301
 - modification rules, 327
 - searching for interactive applications, 60
- Interactive event rule variables, 100
 - event-level, 101
 - form-level, 100
- Interconnect data structure, 52
- Interconnection data structures
 - displaying type definitions, 54
 - modifying form data structures, 52
 - modifying report data structures, 53
- International
 - currency, 255
- Interpretive and compiled code, 296

J

- J.D. Edwards software
 - debugging tools

- interpretive and compiled code, 296
- J.D. Edwards tools
 - Business functions, 149
- Jargon, 28
 - defining, 29
- Java application, 311
- JDB API calls in a business function
 - event rules performance
 - performance tips, 323
- JDBFetch statements
 - event rules performance
 - performance tips, 323
- JDBFetchNext
 - event rules performance
 - performance tips, 323
- jde.ini
 - enterprise server
 - setting, 252
 - record change detection, 254
- jde.ini file
 - setting for transaction processing, 252
- JDEBase, 254
- JDEBASE API
 - standard categories, 172
- JDECACHE, 219
 - initializing the cache, 223
 - JDECACHE standards, 233
 - jdeCacheInit/jdeCache terminate rule, 225
 - JDEKRN, 216
 - multiple cursor support, 231
 - partial keys, 232
 - performance considerations, 217
 - setting up indices, 220
 - using an index to access the cache, 224
 - using in multiple business functions or forms, 225
 - when to use JDECACHE, 216
- JDECACHE API set, 217
 - JDECACHE management APIs, 217
 - JDECACHE manipulation APIs, 218
- JDECACHE cursor
 - how a JDECACHE partial key works, 233
- JDECACHE cursors, 226
 - closing a cursor, 231
 - deleting records, 230
 - HJDECURSOR, 227
 - JDECACHE dataset, 227
 - JDECACHE multiple cursor support, 231
 - JDECACHE partial keys, 232
 - jdeCacheFetchPosition, 230
 - jdeCacheFetchPositionByRef, 230
 - opening a JDECACHE cursor, 226

- resetting the cursor, 231
- types of JDECACHE cursors APIs
 - cursor-advancing APIs, 229
 - non-cursor-advancing APIs, 229
- updating records, 229
- JDECACHE dataset, 227
- JDECACHE management APIs, 217
- JDECACHE manipulation
 - APIs, 218
- JDECACHE manipulation APIs, 218
- JDECACHE multiple cursor support, 231
- JDECACHE partial keys, 232
- JDECACHE standards, 233
 - cache business function description, 234
 - cache business function source description, 233
 - cache business function source name, 233
- jdeCacheDelete, 230
- jdeCacheFetchPosition, 229
- jdeCacheInit, 223
- jdeCacheInit/jdeCacheTerminate rule, 225
- JDEDATE
 - data type, 170
- JDEKRN, 216
- Joined business view versus table I/O
 - business view performance
 - performance tips, 320
- Joined views
 - naming conventions, 44

K

- Key column violation, 318

L

- Language
 - alternate language terms for data items, 28
- Language considerations for processing options, 243
- Languages
 - updating data items, 30
- Last grid rec has been read
 - runtime processing, 84
- Level break messages, 276
- Level messages, 263
- Level-break messages, 276
 - creating a data dictionary item, 279
 - data dictionary
 - creating a data structure, 280
 - first-level messages, 277
 - second-level messages, 277
 - third-level messages, 277

- Link section
 - build output, 184
- Linking
 - business function objects, 181
- Literal values
 - event rule logic
 - defining, 100
- Local window, 304
- Locating an existing error message, 270
- Logic
 - event rule
 - printing, 95

M

- Makefile section
 - build output, 183
- Manipulation
 - event rule, 70
- Manual error settings, 265
- Master business functions
 - creating
 - processing options, 190
 - information structure
 - standard parameters, 207
 - master file, 91, 205
 - naming convention, 190
- Master file
 - master business functions, 91, 205
- Math_Numeric
 - currency
 - application, 257
 - currency implementation, 256
- MATH_Numeric
 - data type, 169
- Media object data structures, 52
 - creating a media object data structure, 55
 - media object storage - F00165 table, 55
- Media object storage - F00165, 55
- Memory allocation
 - business function performance, 324
- Menu
 - Data Dictionary menu - GH951, 17
- Message types, 262
 - error messages, 263
 - information messages, 263
 - level messages, 263
- Messages
 - action messages, 278
 - attaching a data structure template, 272
 - batch error messages, 276
 - defining an active message, 275
 - first-level messages, 277

- level break messages, 276
- level-break messages, 276
- second-level messages, 277
- text substitution error messages, 277
- third-level messages, 277
- workflow messages, 263
- Messaging, 262
 - batch error messages, 276
- Microsoft Visual C++
 - customizing the environment, 305
 - debugging business functions, 301
 - features, 303
 - go command, 303
 - local window, 304
 - setting breakpoints, 304
 - step command, 303
 - step into command, 303
 - using watch, 304
- Modal form interconnect
 - creating, 119
- Modeless form interconnect
 - creating, 120
- Modification rules
 - application text, 331
 - business functions, 335
 - business views, 332
 - control tables, 332
 - data structures, 333
 - event rules, 335
 - interactive applications, 327
 - PeopleSoft EnterpriseOne software, 327
 - report specifications, 330
 - reports, 330
 - table specifications, 331
 - versions, 335
- Modifying form data structures, 52
 - available objects tab, 53
- Modifying report data structures, 53
- Multilevel error messaging for C business functions, 268
- Multi-level error messaging for C business functions
 - create multi-level error messaging, 269
- Multiple-table or single-table
 - business view performance
 - performance tips, 319

N

- Naming
 - transaction master business function modules, 191
 - begin document, 192

- Cancel document (optional), 199
- clear cache, 198
- Edit document, 196
- edit line, 194
- end document, 197
- Naming a data item, 20
- Naming convention, 208
 - master business functions, 190
- Naming conventions
 - data dictionary items, 19
 - data item alias, 19
 - external data dictionary item, 19
 - data item name, 18
 - external data dictionary item, 18
 - joined views, 44
- Naming data items, 15
- Naming standards
 - guidelines when adding a business view, 43
- Native locking strategy, 253
- Next number trigger, 25
- Next Numbers table - F0002, 25
- Node
 - expanded
 - BrowsER, 145
- Non-cursor-advancing APIs, 229

O

- Object browse window, 297
- Object ID
 - show
 - BrowsER, 145
- Object librarian, 9
- Objects, 5
 - searching for objects, 59
 - storing objects, 8
- Objects and applications, 5
 - application, 6
 - object, 5
- ODBC - Open database connectivity
 - PeopleSoft EnterpriseOne, 171
- Open database connectivity
 - PeopleSoft EnterpriseOne, 171
- Opening a JDECACHE cursor, 226
- Optimistic locking, 254
 - database locking, 254
- Options
 - BrowsER
 - working with, 145
- Oracle, 317
- Overrides, 315
- Overview of the debugging process, 291

P

- Page-at-a-time processing
 - runtime processing, 77
- Parameter
 - creating
 - documentation, 211
- Parent DLL
 - business functions
 - changing, 178
- Paste
 - event rules, 94
- Paste options
 - setting, 94
- PeopleSoft EnterpriseOne
 - ODBC - Open database connectivity, 171
 - open database connectivity, 171
- PeopleSoft EnterpriseOne software
 - modification rules, 327
 - specific data type, 168
- PeopleSoft EnterpriseOne software currency
 - implementation, 255
- PeopleSoft EnterpriseOne standards
 - processing option naming standards
 - comment, 242
 - data item, 243
 - processing option data structure, 241
 - tab title, 241
- PeopleSoft EnterpriseOne tools
 - building an application, 7
 - business function design, 11
 - data dictionary, 9, 13
 - event rules, 11
 - form design, 10
 - object librarian, 9
 - processing options, 12
 - report design, 10
 - table design, 31, 33
- PeopleSoft EnterpriseOne Tools, 4
 - business view design, 40
- Performance, 315
- Performance considerations
 - JDECACHE guidelines, 217
- Performance impact
 - format tables, 208
- Performance tips
 - batch applicaiton performance
 - slow performance, 322
 - batch application performance, 322
 - business function performance, 323
 - balance table opens and closes, 324
 - memory allocation, 324
 - business view performance, 319
 - joined business view versus table I/O, 320
 - restrict the number of fields, 320
 - single-table or multiple-table, 319
 - table I/O versus joined business view, 320
 - unions, 320
 - data dictionary performance
 - overrides, 315
 - triggers, 315
 - validation, 315
 - data selection and sequencing, 321
 - data structure performance, 320
 - data structure objects, 320
 - restrict the number of fields, 320
 - error messaging performance, 325
 - event rules performance, 322
 - I/O commands in event rule code, 323
 - JDB API calls in a business function, 323
 - JDBFetch statements, 323
 - JDBFetchNext, 323
 - jdeCache instead of work tables, 323
 - form design, 321
 - find/browse, 321
 - header detail and headerless detail, 322
 - index limitations for various databases, 316
 - Access 32, 316
 - DB2 for OS/400, 317
 - key column violation, 318
 - Oracle, 317
 - specification file corruption, 318
 - SQLSERVER, 317
 - table design performance, 315
 - index considerations, 316
 - table I/O objects, 319
 - transaction processing performance, 325
- Performing
 - build all, 185
- Pessimistic, 254
- Pessimistic record locking
 - business functions, 255
 - transaction boundary, 255
- Portability and standards
 - understanding, 171
- Post dialog
 - initialized, 74
- Pre dialog
 - initialized, 73
- Previewing tables, 35
- Printing
 - event rule logic, 95
- Processing
 - add button, 87

- available objects, 70
- delete button, 89
- runtime, 68
- select button, 85
- Processing option
 - batch application, 238
- Processing option data structure, 241
- Processing option template
 - change a processing option template for text translation, 243
- Processing options, 12, 238
 - add a processing template with translated text, 243
 - attaching a processing options template, 243
 - data structure, 240
 - EnterpriseOne processing option naming standards, 241
 - interactive application, 238
 - language considerations, 243
 - master business functions
 - creating, 190
 - PeopleSoft EnterpriseOne processing option naming standards
 - comment, 242
 - data item, 243
 - processing option data structure, 241
 - tab title, 241
 - templates, 238
- Processing options data structures, 52, 56
- Processing work center APIs
 - calling the processing work center APIs, 288
- Program
 - application encountering errors, 306
 - debug logs, 307
 - ending unexpectedly, 306
 - function being called as expected, 307
 - output of the program incorrect, 307
 - source of the problem, 307
 - SQL log tracing, 308
- Programming standards
 - cache programming standards, 234
 - cache action code standards, 235
 - cache name, 235
 - data structure standard data items, 235
 - defining the cache data structure, 235
 - general standards, 234
 - terminating versus clearing cache, 234

R

- Reading
 - build output, 183
- Rebase section

- build output, 184
- Rebuilding cross reference information, 64
- Record, 254
- Record locking, 253, 254
 - native locking strategy, 253
 - optimistic locking, 254
- Records
 - deleting records, 230
 - jdeCacheDelete, 230
 - jdeCacheFetchPosition, 229
 - updating records, 229
- Related functions
 - adding, 178
- Related tables and functions
 - adding, 177
- Removing tables from the database, 37
- Replicated data dictionary
 - storing data dictionary and data dictionary items, 15
- Report
 - defining transaction processing, 252
- Report data structures, 51
 - modifying report data structures, 53
- Report design, 10
- Report specification modification rules, 330
- Resetting errors, 267
- Resetting the cursor, 231
- Restrict the number of fields
 - business view performance
 - performance tips, 320
 - data structure performance
 - performance tips, 320
- Reviewing
 - error output, 189
- Rollback, 245
- Row
 - exited and changed - asynch, 140
- Row and column text
 - change for all applications, 30
- Runtime
 - data structures, 68
 - processing, 68
- Runtime data structures and available objects, 68
- Runtime message
 - defining the glossary text, 272
- Runtime processing
 - all grid recs deleted from DB, 91
 - BC assigned database values, 77
 - button clicked, 86
 - delete grid rec from DB after, 91
 - delete grid rec from DB before, 91
 - delete grid rec verity after, 90

- delete grid rec verity before, 90
- do in while loop, 77
- grid rec is fetched, 78
- last grid rec has been read, 84
- page-at-a-time processing, 77
- SQL select, 76
- write grid line after, 82
- write grid line before, 80

S

- Saving and creating
 - event rules, 92
- Search
 - BrowsER, 145
- Search combo box, 299
- Searching for batch applications, 60
- Searching for business functions, 61
- Searching for business views, 61
- Searching for data items, 59
- Searching for data structures, 62
- Searching for event rules, 63
- Searching for forms, 62
- Searching for interactive applications, 60
- Searching for objects, 59
- Searching for tables, 62
- Second-level messages, 277
- Security
 - universal table browser, 37
- Select button
 - processing, 85
- Select distinct, 42, 47
- Send Message system function, 273
- Send message system functions
 - defining an active message, 275
 - use the send message system function, 274
- Setting
 - build options, 181
 - enterprise server
 - jde.ini, 252
 - paste options, 94
- Setting breakpoints, 301, 304
- Setting up currency conversion, 258
- Show
 - object ID
 - BrowsER, 145
- Showing currency-sensitive controls, 258
- Simple error messages, 271
 - adding an interactive message data item, 271
- Single record processing
 - transaction master business function
 - implementing, 203
- Single-table or multiple-table

- business view performance
 - performance tips, 319
- Size of alias
 - for use in an RPG program, 19
- Slow performance, 322
- Smart field trigger, 26
- Special logic/processing required
 - begin document, 192, 195
 - cancel document (optional), 199
 - clear cache, 198
- Specification file corruption, 318
- SQL log tracing, 308
- SQL select
 - Runtime processing, 76
- SQLSERVER, 317
- Standard
 - JDEBASE API categories, 172
- Standard parameters, 207
- Standards
 - cache programming standards, 234
 - cache action code standards, 235
 - cache name, 235
 - data structure standard data items, 235
 - defining the cache data structure, 235
 - general standards, 234
 - terminating versus clearing cache, 234
 - JDECACHE standards, 233
 - cache business function description, 234
 - cache business function source description, 233
 - cache business function source name, 233
- Standards and portability
 - understanding, 171
- Step command, 303
- Step into command, 303
- Steps
 - database communication
 - understanding, 173
- Storing objects
 - central-storage server, 8
 - workstation or server, 8
- Structure
 - business function source files, 163
 - cache, 199
 - detail, 201
 - header, 200
- Structures
 - runtime, 68
- Summary section
 - build output, 185
- System function
 - event

- attaching, 103
- System function tracing, 309
- System functions
 - system function tracing, 309
 - transaction processing, 248
- System Functions
 - send message system functions, 273
- System generated data structures, 51
 - form data structures, 51
 - report data structures, 51

T

- Tab title, 241
 - defining a tab title for a processing option, 241
- Table
 - adding a table
 - indices, 33
 - choosing a table for a business view, 45
 - choosing data items, 34
 - event rules, 67
 - Next Numbers - F0002, 25
- Table design, 9, 31, 33
 - choosing data items for the table, 34
 - column properties, 39
 - defining indices, 34
 - index, 31
 - previewing tables, 35
 - tables, 31
 - universal table browser
 - formatted data, 39
 - unformatted data, 38
 - viewing data in tables, 37
- Table design performance, 315
 - index considerations, 316
- Table event rules, 12, 135
 - creating, 135
- Table I/O
 - available operations, 128
 - event rule
 - creating, 129
 - event rule variable, 102
 - Event Rules Design, 127
 - extending a transaction boundary, 251
 - valid mapping operators, 128
- Table I/O objects, 319
- Table I/O versus joined business view
 - business view performance
 - performance tips, 320
- Table join, 40
 - creating a table join, 50
- Table specifications modification rules, 331

- Table union, 41
 - creating a table union, 51
- Tables
 - copying tables, 36
 - defined, 31
 - deleting, 37
 - generating header files, 36
 - generating indexes, 36
 - generating tables, 35
 - previewing tables, 35
 - removing from database, 37
 - searching for tables, 62
 - viewing data in tables, 37
- Template
 - business function documentation, 210
 - data structure documentation, 210
- Templates
 - processing options, 238
- Terminating the work center process, 290
- Terminating versus clearing cache, 234
- Text functions
 - advanced functions
 - expression manager, 126
- Text substitution error messages, 277
- Text translation
 - change a processing option template, 243
- The JDECACHE API set, 217
- The jdeCacheFetchPosition API, 230
- The jdeCacheFetchPositionByRef API, 230
- Third-level messages, 277
- Threaded events
 - asynchronous events, 138
- Tips of the day, 260
- Transaction boundary, 246
 - data interdependence, 246
 - extending a transaction boundary using table I/O, 251
 - extending between forms, 250
 - extending using business functions, 251
 - pessimistic recording locking within transaction boundary, 255
- Transaction master business function
 - Components, 191
- Transaction master business function modules
 - naming, 191
 - begin document, 192
 - cancel document (optional), 199
 - clear cache, 198
 - Edit document, 196
 - edit line, 194
 - end document, 197
- Transaction master business functions, 190

- building, 202
- implementing, 203
 - document processing, 204
 - single record processing, 203
- Transaction processing, 244, 245, 249
 - commits and rollbacks, 245
 - data interdependence, 246
 - defining for a report, 252
 - defining transaction processing for a form, 249
 - extending a transaction boundary, 250
 - form types, 249
 - remote business functions, 248
 - system functions, 248
 - transaction boundaries, 246
- Transaction processing and business functions, 248
- Transaction processing in remote business functions, 248
- Transaction processing performance, 325
- Transaction processing scenarios, 246
- Transaction processing system functions, 248
- Translated text
 - adding a processing template with translated text, 243
- Trig functions
 - advanced functions
 - expression manager, 125
- Trigger
 - build triggers option, 256
 - creating a currency conversion trigger, 259
- Triggers, 315
 - attaching default triggers, 21
 - default triggers, 16
 - default value trigger, 22
 - display rule trigger, 24
 - edit rule trigger, 23
 - next number trigger, 25
 - smart field trigger, 26
 - user defined codes trigger, 25
 - visual assist trigger, 22
- Turning on debug tracing, 309
- Two-phase commit (manual commit mode), 245
- Type definitions
 - displaying type definitions, 54
- Typical
 - event flow
 - Find/Browse form, 72
- Typical users and hook up
 - begin document, 195

U

- Understanding
 - business function builder form, 180
 - database communications steps, 173
 - event rules, 66
 - event rules buttons, 67
 - portability and standards, 171
 - standards and portability, 171
- Understanding default triggers, 16
- Understanding error handling, 264
- Understanding how level break messages work, 276
- Understanding how PeopleSoft EnterpriseOne stores objects, 8
- Understanding PeopleSoft EnterpriseOne software modification rules, 327
- Understanding record locking, 254
- Understanding the development cycle, 7
- Understanding the event rules debugger, 297
- Understanding transaction processing, 245
- Unformatted data, 38
- Unions
 - business view performance
 - performance tips, 320
- Universal table browser
 - formatted data, 39
 - unformatted data, 38
 - using security in the universal table browser, 37
 - viewing data in tables, 37
- Updating data items for languages, 30
- Updating records, 229
- Updating the glossary, 28
 - adding glossary text for languages, 28
- Useful features of the Visual C++ debugger, 303
- User defined codes trigger, 25
- User generated data structures, 52
 - media object data structures, 52
 - processing options data structures, 52
- Using
 - database APIs, 170
 - utility programs, 182
- Using an index to access the cache, 224
- Using pessimistic record locking within a transaction boundary, 255
- Using security in the universal table browser, 37
- Using select distinct, 47
- Using the data dictionary, 17
- Using the jdeCacheInit/jdeCacheTerminate rule, 225
- Using the same cache in multiple business functions or forms, 225
- Using watch, 304

Utility programs
using, 182

V

Valid mapping operators
Table I/O, 128

Validating
event rules, 96

Validation, 315

Value
assign, 123

Variable selection and display window, 298

Variables
Grid, 101

Versions modification rules, 335

Viewing
documentation
business function - values to pass, 213
business function document viewer, 214
business function search, 213

Viewing field relationships, 63

Viewing the data in tables, 37

Visual assist trigger, 22

Visual ER Compare, 146

W

Web-based applications
required components, 311
Java application, 311

While statements

event rule logic, 97

Work center API, 278

Work center initialization API
calling the work center initialization API, 287

Work center process
terminating the work center process, 290

Workflow messages, 263

Working with
BrowsER, 144

BrowsER options, 145

business function builder, 179

Working with asynchronous processing, 138

Working with business view design, 44

Working with currency, 256

Working with error messages, 270

Working with table design, 33

Working with the Data Reference Scan, 336

Working with the event rules debugger, 297

Working with the Send Message System
Function, 273

Working with the Tip of the Day utility, 260

Working with the Visual C++ Debugger, 303

Working with tips of the day, 260

Working with transaction processing, 249

Workstation jde.ini
record change detection, 254

Write grid line after
runtime processing, 82

Write grid line before
runtime processing, 80

